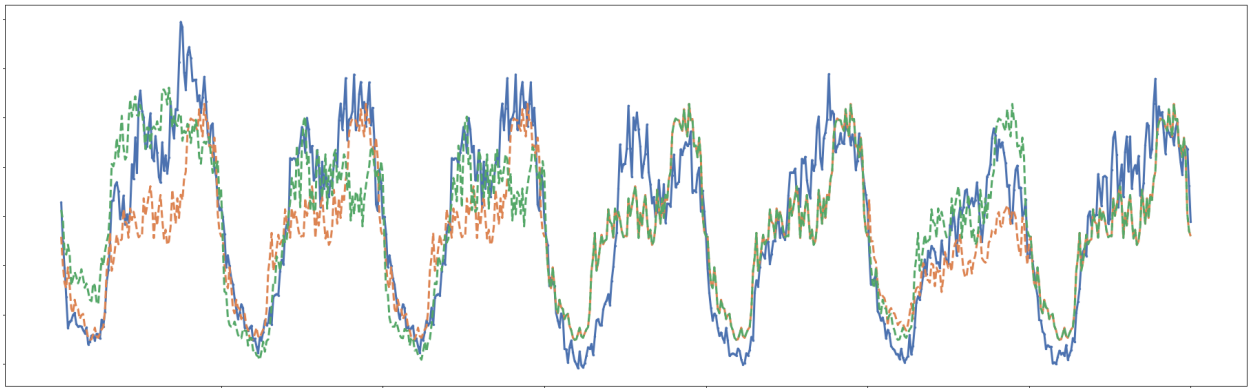


Compressed Machine Learning on Time Series Data

Efficient compression through clustering using candidate selection and the application of machine learning on compressed data

Master's thesis in Computer science and Engineering

FELIX FINGER
NATHALIE GOCHT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Compressed Machine Learning on Time Series Data

Efficient compression through clustering using candidate selection and the application of machine learning on compressed data

FELIX FINGER
NATHALIE GOCHT

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Large-scale Clustering of Time Series
Efficient compression through clustering using candidate selection and the application of machine learning on compressed data

FELIX FINGER, NATHALIE GOCHT

Supervisor: Alexander Schliep, Computer Science & Engineering
Advisor: Gabriel Alpsten and Sima Shahsavari, Ericsson
Examiner: Devdatt Dubhashi, Computer Science & Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Prediction using compression

Typeset in LaTeX
Gothenburg, Sweden 2020

Large-scale Clustering of Time Series
Efficient compression through clustering using candidate selection and the application of machine learning on compressed data
FELIX FINGER
NATHALIE GOCHT
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The extent of time related data across many fields has led to substantial interest in the analysis of time series. This interest meets growing challenges to store and process data. While the data is collected at an exponential rate, advancements in processing units are slowing down. Therefore, active research is practiced to find more efficient means of storing and processing data. This can be especially difficult for time series due to their various shapes and scales.
In this thesis, we present two variants for optimising a Greedy Clustering algorithm used for lossy time series compression. This study investigates, whether the efficient but lossy compression sufficiently preserves the characteristics of the time series to allow time series prediction and anomaly detection. We suggest two variants for a performance optimization, *Greedy SF* and *Greedy SAX*. These algorithms are based on novel lookup methods for cluster candidate selection based on statistical features of time series and extracted SAX substrings. Furthermore, we enabled the clustering to allow processing time series with different value ranges, which allows the compression of time series with various scales. To validate the end-to-end pipeline including compression and prediction, a performance evaluation is applied. To further analyse the applicability, a comprehensive benchmark against a pipeline with an autoencoder for compression and a stacked LSTM for prediction is performed.

Keywords: time series clustering, large scale data, machine learning, prediction, anomaly detection, compression

Acknowledgements

We want to express our gratitude towards our academic supervisor Alexander Schliep and our industrial supervisors Gabriel Alpsten (Ericsson) and Sima Shahsavari (Ericsson) for their continuous, useful feedback and guidance throughout this project. Especially during complicated circumstances regarding the corona virus, we appreciate the support even more. In addition, a thank you to Bengt Sjögren for helping us getting familiar with spark and supporting us through out our challenges. We would also like to acknowledge our examiner Devdatt Dubhashi for his helpful feedback and remarks during the midterm discussion.

Felix Finger and Nathalie Gocht, Gothenburg, June 2020

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

APCA	Adaptive Piecewise Constant Approximation.
DFT	Discrete Fourier Transform.
DWT	Discrete Wavelet Transform.
ED	Euclidean Distance.
LSTM	Long Short Term Memory.
PAA	Piecewise Aggregate Approximation.
RNN	Recurrent Neural Network.
SAX	Symbolic Aggregate ApproXimation.
SF	Statistical Features.

Glossary

cluster candidate	A subset of already computed prototypes that are returned by the method getCandidates during the greedy clustering algorithm..
cluster label	Each cluster representative can be addressed with its label which corresponds to the index of the representative in the list of cluster centers.
cluster representative	For every cluster computed in the greedy clustering algorithm, there is a cluster centroid, which contains the time series of 96 data points, that was used to create the cluster. This centroid is also called cluster representative.
Greedy Plain	Greedy clustering without candidate selection.
Greedy SAX	Greedy clustering improved by candidate selection using SAX sub strings.
Greedy SF	Greedy clustering improved by candidate selection using statistical features.

1

Introduction

There are many disciplines where data has a temporal component. This could involve natural processes like weather or sound waves and human-made processes, like a financial index or sensor data of a robot. In general, time series are collected and analysed in many areas of science such as in astronomy, biology, meteorology, medicine, finance, robotics and engineering. This collected data has been growing exponentially in the last years and it is getting more difficult to store and process this information [20]. These challenges increase for machine learning applications on large data sets due to their computational complexity. Data compression might be necessary to enable machine learning models. However, compressing time series is a challenge of its own because of their noisy and complex nature. One way of compressing time series is clustering. Clustering is used to compress time series by finding cluster representatives for similar groups of time series. Active research has been going on and is continuously done in compressing and utilizing the information contained in time series by means of clustering algorithms. As of today, clustering on time series is still not an efficient process and requires a lot of computing time. This thesis work introduces an improved version of the Greedy Clustering algorithm proposed by [2] to dynamically handle different scales in the time series. Moreover, two novel lookup methods for cluster candidate selection based on time series statistics and SAX features are suggested to reduce the complexity of this algorithm. Finally, a data set provided by Ericsson's radio network stations is compressed by this algorithm and used for evaluating whether time series prediction and anomaly detection can be applied.

1.1 Background

Time series can be compressed using different methods. This study focuses on a clustering-based method. Alternative solutions are described in related work. There are various clustering algorithms. Some are not especially applicable for time series such as k-means and others are specialised for time series data such as k-shape. These algorithms have a crucial disadvantage when dealing with large amounts of data as they require multiple iterations until convergence. These algorithms cluster by comparing every time series in every iteration with every cluster. They belong to the class of NP hard problems and are therefore not applicable when hardware is restricted and amount of data is large. Using k-means, following Lloyd's algorithm,

for time series clustering has a complexity of $O(i \times c \times n \times l)$, where i is the number of iterations, c the number of clusters, n the size of the data set and l the length of a time series [9].

However, there is a clustering algorithm that only iterates twice over the data set, called the Greedy Clustering algorithm. This algorithm creates cluster in the first iteration and performs possible reassignments in the second iteration. In every iteration, the series is compared to every cluster during the clustering, a total of $n \times c$ comparisons is needed. Therefore, it is crucial to employ some technique so that the number of comparisons can be reduced [2]. Essentially, the set of potential clusters a time series can be assigned to needs to be narrowed down to decrease the number of comparisons. This accounts especially for time series with an irregular distribution. One downside of this algorithm is its focus on efficiency rather than cluster quality. To further maintain precision, the settings of the clustering need to be adjusted, such that approximately 1-10% of the time series are converted to clusters. Considering an example of 1 million time series and 100 000 clusters, the complexity would be very high even for this algorithm due to its large number of similarity comparisons. Thus, we counter this situation by implementing two lookup tables utilizing two kinds of time series features to decrease the number of cluster comparisons during the clustering. There is a chance that similar time series will not have similar enough features to be found by the lookup table. Hence, it does not guarantee finding the most similar matches, but at least finding somewhat similar matches given the computational resources. The efficient clustering of large quantities of time series could enable the application of machine learning tasks as anomaly detection or prediction. We assume that a sequence prediction could be applied on the cluster labels instead of the original time series in an efficient manner while maintaining an acceptable loss.

Based on these assumptions, we identified two research questions.

Question 1: How and to what extent can the computational complexity in time and space, of existing time series clustering methods, be further decreased?

Question 2: How can machine learning techniques for the prediction of future time series and anomaly detection on the compressed time series be designed and implemented?

1.2 Related Works

There are many known methods for data compression. Salomon and Motta [21] state that even though they are based on different ideas, are suitable for different categories of data, they are all based on the same principle of compressing data by removing redundancies. On the one hand, time series can be compressed lossless in byte form using standard compression techniques as e.g. Snappy [8] introduced by Google or Zstd [5] presented by Facebook. On the other hand, they can be compressed lossy using different methods. Clustering can be used to remove redundant

information by finding common structures in time series and only storing the cluster representative. Another possibility for compression are neural networks that could be trained on the time series to find a dense representation of the data by projecting it into lower space. This class of neural networks belongs to the category of autoencoders.

Our master thesis is inspired by Alpsten and Sabi's master thesis "Prototype-based compression of time series from telecommunication data [2]. They implemented a new type of compression for time series based on clustering and residuals with the aim to compress the data storage of time series. For the evaluation of the clustering, they used data sets with less than 100 000 time series. Different algorithms were analysed and one promising algorithm in terms of computational efficiency was the Greedy Clustering algorithm presented in Section 2.1.2.

Our focus of this study is enabling machine learning on large time series data sets. When dealing with large data sets, lossy compression is beneficial not only in terms of data storage but also due to smaller features as input for machine learning models. We identified a research gap here, as clustering of time series is still not an efficient process where 1 - 10% of the data set can be translated to clusters. Therefore, we used the Greedy Clustering algorithm proposed by Alpsten and Sabi as a starting point for improvement in regard to larger data sets.

For further background, a broad survey on data mining of time series data is given by Liao [23]. In this paper, various time series clustering approaches are presented and described.

In order to improve the efficiency of the Greedy Clustering algorithm, two types of candidate selections are implemented and compared with the plain Greedy Clustering. One candidate selection is based on the Symbolic Aggregate ApproXimation (SAX) representation as presented in [14]. The SAX representation is often compared to other time series representations such as Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT) and Adaptive Piecewise Constant Approximation (APCA) [13]. An alternative to SAX is introduced by Malinowski and Guyet et al. [15], who presents the 1d-SAX representation that takes the slope of each segment of the Piecewise Aggregate Approximation (PAA) into account. Sirisambhand and Ratanamahatana [22] propose a dimensionality reduction technique for time series data by combining the PAA segmentation with an Additive Representation.

Another related project using SAX feature extraction to enable indexing and mining of a very large number of time series was presented by Camerra and Palpanas et al. They implemented a novel tree based index structure iSAX 2.0 [4].

1.3 Roadmap

The required theoretical knowledge for the techniques used in clustering, machine learning and evaluation are presented in Section 2. This section is followed by a detailed description of the methods used. Figure 1.1 guides the reader in the

general process of this research and visualizes the main concepts described in detail in Section 3. This section contains information about the data sets, the data retrieval using Spark and data cleaning and smoothing techniques. It describes the two novel algorithms Greedy SF and Greedy SAX and the magnitude adaptive clustering that allows to compress time series of different scale. These methods are followed by the data engineering required for the respective machine learning applications including the cluster based prediction and the benchmark model that predicts values. Finally, it is outlined how the evaluation of the clustering results and the machine learning tasks is applied. The results are presented and discussed in Section 4 and summarized in a conclusion. Finally, different ideas on future work are given.

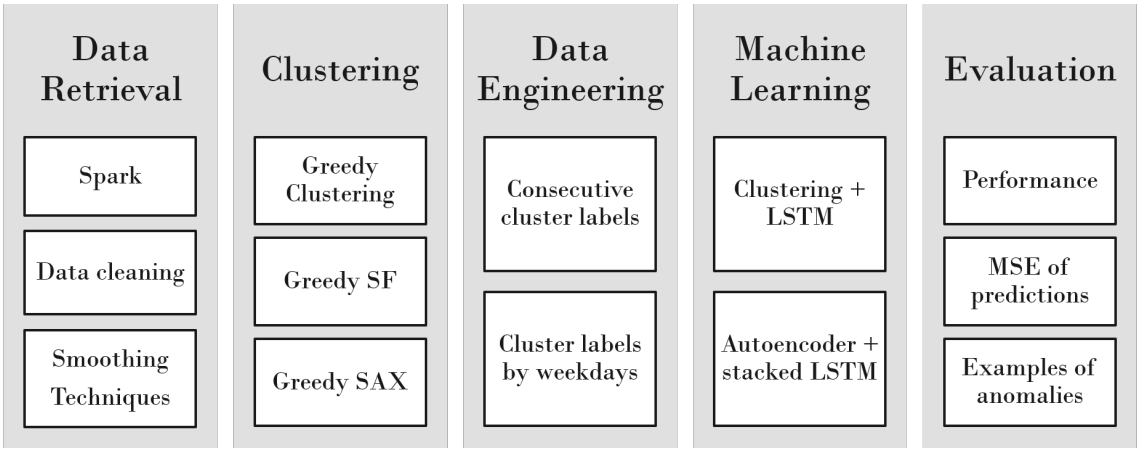


Figure 1.1: Roadmap

1.4 Ethical Considerations

The data is collected from Ericsson's Network management tools and provides information about network performance on the cell level. Hence, the data used within this research does not contain any personal user specific information and does not compromise any user privacy. The results in this thesis could have many positive effects as it enables the application of a wide range of tools that could otherwise not be applied.

2

Theory

2.1 Time Series Clustering

Using clustering, n observations are partitioned into k clusters, where a cluster is characterized with the notions of homogeneity, the similarity of observations within a cluster and the dissimilarity of observations from different clusters. In time series clustering, an observation consists of values measured over a defined time interval[19]. Several methods have been proposed to cluster time series. All approaches generally modify existing algorithms, either by replacing the default distance measures with a version that is more suitable for comparing time series (raw-based methods), or by transforming the sequences into flat data, so that they can be directly used in classic algorithms (feature- and model-based methods)[23]. The Greedy Clustering in this research is a centroid based algorithm and Figure 2.1 demonstrates an example of a centroid in red and its assigned cluster members in grey.

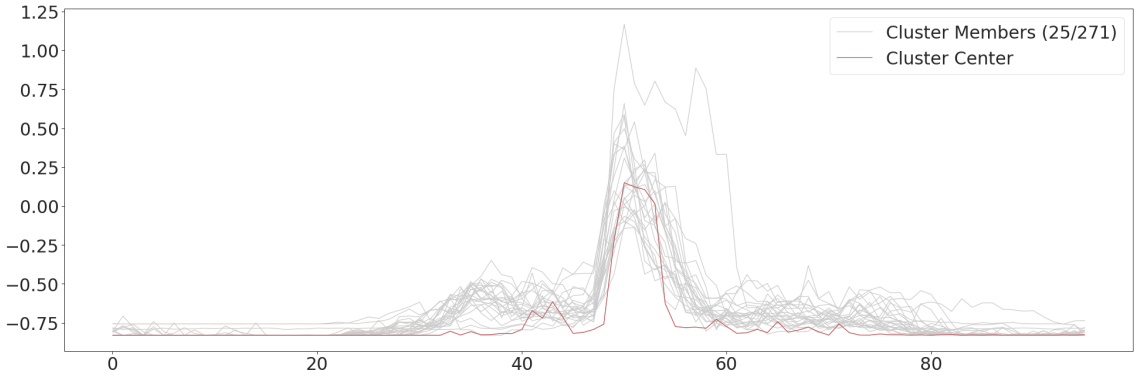


Figure 2.1: Example cluster with its cluster members.

2.1.1 Normalization and Similarity Measures for Time Series

In order to prepare time series for clustering, they need to be normalized as differences in scale can lead to false similarity comparisons. In general, approaches for time-series comparison use the z-score normalization. A distance measure is then used to determine their similarity and captures possibly more invariances[19]. The

z-normalization is defined as follows, where μ is the mean and σ is the standard deviation over the entire normalized data set,

$$z = x - \frac{\mu}{\sigma}. \tag{2.1}$$

By keeping the values for μ and σ used for normalization for every data set, the initial values can be restored afterwards.

Another option to normalize the data is the min-max normalization. This scales all time series values between zero and one and is defined as in[1]

$$x' = \frac{x - \min}{\max - \min}. \tag{2.2}$$

The distance metric used in this research is the euclidean distance (ED)[7]. It is defined as follows, where x and y correspond to vectors of the time series and n represents the length of the time series,

$$ED(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2.3}$$

The computation of the ED distance is efficient and the distance measure penalizes large deviations as squares grow faster, which applies to small numbers and therefore works on normalized data. Here, value 0 would indicate that the time series are exactly equal to each other. The comparison performed by the euclidean distance may not catch all the similarities as time series could be not perfectly aligned. In this case the distance could become very large despite a very similar shape.

2.1.2 Greedy Plain Clustering

Algorithm 1 shows the Greedy Clustering algorithm as it was presented by Alpsten and Sabis[2]. In this research we will refer to this algorithm as Greedy Plain. Compared to other clustering algorithms as k-means or k-shape, the Greedy Clustering method only works as a two-pass algorithm and collects cluster centers rather than computing them. Additionally, the resulting clusters will most likely not separate the data set in clearly defined clusters. Since clusters are formed "on the go", overlapping clusters are possible.

The number of clusters k is not specified beforehand in the Greedy Clustering approach. The algorithm has two stages, the cluster formation and the cluster assignment. In the first pass, we iterate through every time series. The first time series forms the first cluster centroid and is also assigned as a cluster member. For every new time series, its distances to the existing cluster centers are calculated. If the shortest distance exceeds a specified threshold *tau* (τ), the time series forms a new

Algorithm 1: Greedy Clustering (Baseline)

Input : dataset, tau, distance measure
Output: cluster assignments, clusters
Cluster formation:

```
cluster = [ ];
for each time series ts do
    if there exist no clusters c so that d(c,ts) < tau then
        add ts to cluster
```

Cluster assignment:

```
for each time series ts do
    find a clusters c so that d(c,ts) is minimal;
    assign ts to that cluster
return cluster assignments, clusters
```

cluster and becomes the center or representative. If the distance is smaller than *tau*, the time series is assigned to the already existing cluster. Here, *tau* can be defined as the radius around each cluster center. If a data point falls within this radius it belongs to that cluster. By setting *tau* very small, more clusters with fewer members are created. When *tau* is chosen rather large, less cluster will be formed. In the second stage every time series is compared to every cluster to find the final cluster assignment with the smallest distance. The complexity for the second pass is $O(n*k)$ where n is the number of time series and k is the number of clusters created. Taking the complexity of this algorithm into account, it is much less complex than k-means or k-shape.

2.2 Symbolic Aggregate ApproXimation (SAX)

The SAX transformation converts continuous-valued time series in a stream of characters that represents the original time series. Figure 2.2 shows the basics of the SAX transformation. Two parameters are chosen for the transformation: α, the size of the alphabet, and w, the number of words to be produced. First, the time series is normalized and segmented into equal sized user-defined pieces using Piecewise Aggregate Approximation (PAA), which reduces the dimensionality of the time series. Each time series need to be normalized individually to have a mean at zero and a standard deviation of one. The value of each segment is the mean of the data points in that segment. These segments are then translated into symbolic words using a breakpoint table. Breakpoints define numeric borders. If two segments fall above and under a breakpoint they have a different letter assigned. According to Lin et al.[13], breakpoints are defined as a sorted list of $a-1$ numbers that split the area under a $N(0,1)$ Gaussian curve in a equal-sized areas. The number of breakpoints is equal to $\alpha - 1$. The values of the breakpoint can be looked up in a statistical table.

a breakpoint β	3	4	5
β_1	-0.43	-0.67	-0.84
β_2	0.43	0	-0.25
β_3		0.67	0.25
β_4			0.84

Table 2.1: Breakpoint table for $a = 3, 4, 5$ (comp. [13]).

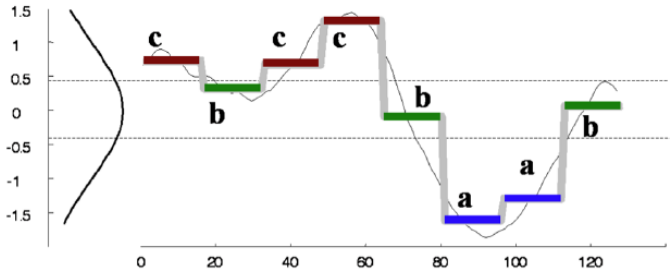


Figure 2.2: Example of SAX for a time series, with parameters $\alpha = 3$ and $w = 8$. The time series above is transformed to the string *cbccbaab*, and the dimensionality is reduced from 128 to 8[14].

Table 2.1 shows the breakpoints for $a = \alpha - 1 = \{3-5\}$. In Figure 2.2 the breakpoint borders are shown as horizontal lines at 0.43 and -0.43. These borders state at which point the segment has a different symbolic word. Due to the normalization of every individual time series, the SAX representation doesn't preserve the scale or distribution of the entire data set but only the shape of the time series.

2.3 Autoencoder

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. Autoencoders compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact learned summary of the input, also called the latent-space representation. In other words, it is trying to learn an approximation to a function, so that output $\hat{x}$ is similar to x[18]. An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code. This can be used for compressing images, as well as time series to accelerate machine learning on reduced data sets.

Figure 2.3 presents an example of the basic workflow of the autoencoder used on time series. Four hyperparameters need to be defined before the training, the code size, representing the number of nodes in the middle layer, the architecture of the hidden layers and a loss function.

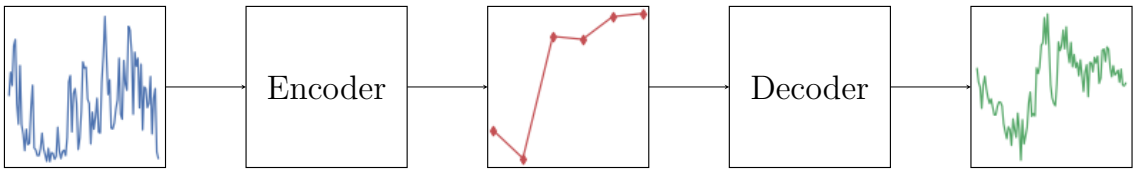


Figure 2.3: Autoencoder workflow.

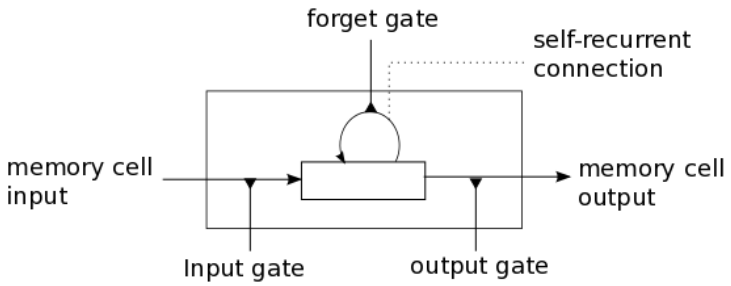


Figure 2.4: Structure of a LSTM cell[6].

2.4 Long Short Term Memory (LSTM)

Recurrent Neural Networks (RNN) can be used to learn and predict sequences since it applies the same function to each input and stores information in an internal memory. One of the general appeals of RNNs is the idea that they are able to connect previous information to the present task. Nevertheless, due to the vanishing gradient problem, RNNs are not able to remember information for a longer period. Therefore, a simple RNN is unable to handle large input sequences[10]. Hochreiter and Schmidhuber proposed the Long Short Term Memory (LSTM) model based on a RNN. LSTMs are capable of learning long-term dependencies in sequences and remembering information for a longer period of time[11].

Figure 2.4 demonstrates the structure of one recurrent unit within the LSTM. In comparison to a simple RNN, it has three gates, an input gate processing the current state, a forget gate that contains information about previous states as well as an output gate that combines the current information with the previous knowledge. The forget gate ensures to forget information that is not important and tries to learn important repeating observations and patterns in sequences.

2.5 Evaluation Metrics

MSE

The Mean Squared Error is commonly used to evaluate regression models. It is defined as the sum over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points. Within this thesis, it is used as a loss function within the autoencoder as well as to measure the error of the predictions. The MSE is

calculated for every predicted cell since cells can respond differently to the prediction method. Therefore, we can evaluate which cells are suitable for the predictions.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.4}$$

Pearson's correlation coefficient

A correlation coefficient r of 1 means that for every positive increase in one variable, there is a positive increase of a fixed proportion in the other. Within this research, the pearson correlation coefficient is used to measure, whether statistical features are useful to infer how close or far away time series are from each other based on the euclidean distance.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.5}$$

3

Methods

This section introduces the data sets used in this thesis, followed by a detailed description of the data retrieval process using a spark pipeline. Then, several smoothing methods for time series are introduced to increase the cluster efficiency. This section is followed by a detailed explanation of the two candidate selections for Greedy SF and Greedy SAX. Subsequently, the magnitude adaptive clustering is outlined, that enables the clustering of time series with different scales, finalized by a description of the clustering evaluation. For the time series prediction, the feature engineering process is outlined including the architectures of the two models in comparison. This section concludes with the model for anomaly detection and how the machine learning results are evaluated.

3.1 Data Introduction

All our mobile devices can be connected to the internet by using radio access network cells. To ensure that these cells have a good performance and stability, performance measurement counters are collected and evaluated. These counters describe different aspects of the ongoing traffic in a cell and store values for every 15 minutes, forming time series of 96 values for every day and cell. We analysed a representative subset of these counters that are interesting to analyse and eventually allow the application of machine learning.

We explored the data sets for different cells in regard to the level of noise and if they show some regular shape or trend. Figure 3.2 and 3.1 present sample time series of different cells for the extracted Counter 47 and 79. On the left, noisy samples are shown, whereas on the right, some trend and regular shape is visible. Regular behaviours among cells can differ due to the cell's location. For example, a cell in a rural area shows a different behaviour than a cell in an industrial area. Another observation is a different behaviour on weekdays compared to the weekend.

If a cell has an uncommon behaviour, it can be an indication that this cell might not be perfectly suited for prediction. This accounts especially for a compressed representation as we have to accept some information loss that could concern these outliers.

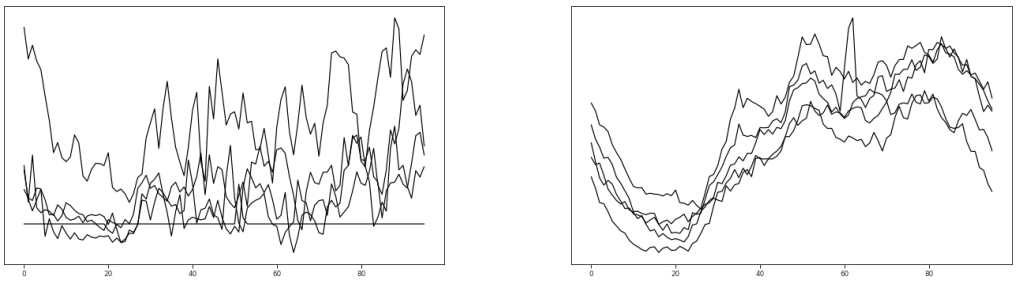


Figure 3.1: Example time series for Counter 79.

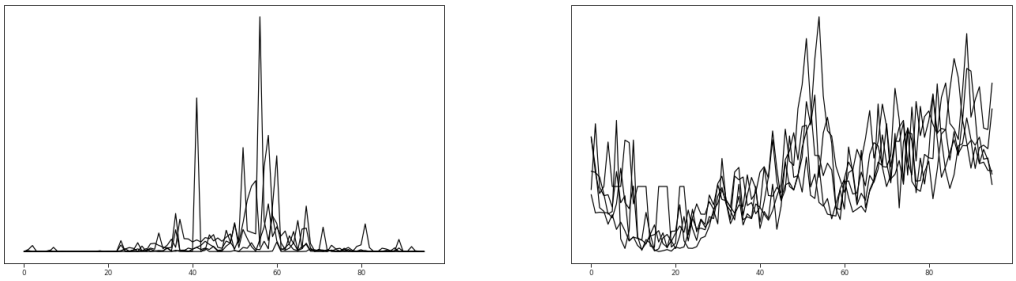


Figure 3.2: Example time series for Counter 47.

This might lead to many different cluster labels for one cell, which could not be used for prediction. On the other hand, for a cell with small variations, models might only predict the same cluster. This type of cell would not be suitable for prediction. Nevertheless, many cells with a common repeating pattern are observable for Counter 79.

Figure 3.3 presents 52 weeks overlapping for one specific cell within the data set of Counter 79. The 672 data points represent the days from Monday to Sunday, where every day consists of 96 data points, respectively. These weeks are visualized in an overlapping fashion, so that differences in behaviour can be observed. Sunday has an observable difference in amplitude and Saturday when compared against the weekdays Monday to Friday. In general, these plots help to evaluate the level of noise within the data, and potential patterns. Some days within the weekdays indicate potential anomalies as the time series deviate from regular behaviour. This supports our assumption that some cells could contain anomalies, allowing the application of anomaly detection. We observed a regular pattern for many cells, which strengthened our assumption that forecasting using compressed time series is possible.

3.2 Data Retrieval

The data and the environment for computation are provided by Ericsson. We used data from a RAN network that is known to deliver good data quality. The data set

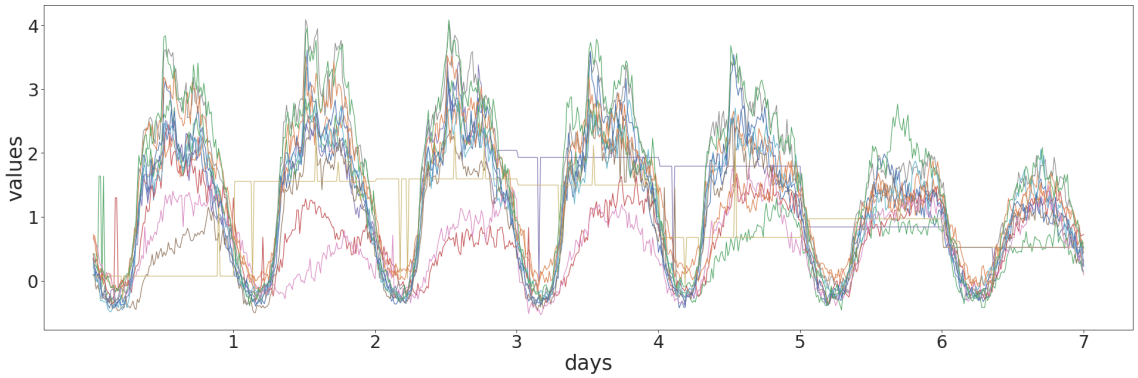


Figure 3.3: Overlapping weeks for one cell of Counter 79. The x-axis represents the data points in a time series. Each week consists of 672 data points. The y-axis shows the normalized values.

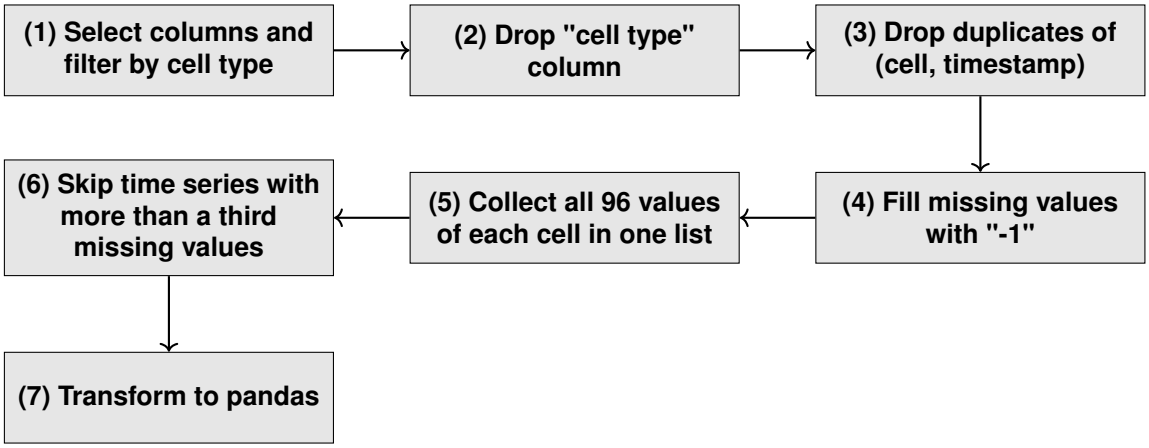


Figure 3.4: Data retrieval workflow using `pyspark`.

is read from parquet files using spark. The workflow of the pyspark logic is presented in Figure 3.4.

We filtered the data set by a specific cell type, where certain Counters are measured. Then, we filter for Counters that are interesting for machine learning applications and at the same time might show a more regular shape (comp. (1) in Fig. 3.4). In the results, only two counters are used. An overview over these two Counters and the purpose within this project is presented in Table 3.1.

Duplicates in cell and timestamp combination are dropped and missing values flagged to be approximated. For every cell, the values are collected in one list by using the `pyspark` function `collect_list` over a window. This window is defined by a `partitionBy` cell and ordered by timestamp. This guarantees that all values of one cell are collected in the order of the timestamps.

Additionally, missing values are approximated with the median of the time series when more than 64 out of 96 data points in a time series are filled. Otherwise the time series is disregarded. This way we can ensure that every time series is complete,

Counter	Data set	GB	Size	Cells	Application
79	Y	4.700	1 016 489	>5 000	clustering
	X	0.943	411 160	>1 000	clustering, machine learning
47	X	1.184	342 740	>1 000	clustering

Table 3.1: Data sets for clustering and machine learning. For each data set the size in Giga Byte, the number of time series and the number of cells is given.

which prevents inconsistencies during the clustering. The data is processed day by day resulting in a spark data frame in which each line represents a cell at this specific date. The columns are the counters which hold arrays of 96 data points for each cell. As a last step, the spark frame is converted in a pandas data frame which is considerably smaller than the raw parquet format. From this pandas data frame, time series objects are created and hold the following information.

- cell name
- date
- raw Counter data
- Counter data z-normalized
- Counter data min-max scaled
- statistical features for both normalisations
- assigned cluster label
- Counter name

The time series are normalized using the z-normalization and the Min-Max scaler described in Section 2.1.1.

Making use of the distributed computing of spark gave us a remarkable speed up while reading and processing the data.

3.3 Smoothing Techniques

Smoothing techniques present an option to reduce noise. For telecommunication data as ours, it can be used to aggregate time series to a common behaviour before compressing the data. This could potentially improve the compression.

Moving Average

"A moving average is defined as an average of fixed number of items in the time series which move through the series by dropping the top items of the previous averaged group and adding the next in each successive average."[17, ch. 12]

It can be used to smooth time series and eliminate the noise, given a window size and sliding window step. The time series in Figure 3.5 consists of 29 280 data points, where every 96 values represent one day. The blue line indicates the original time series for the cell. The red line shows the basic moving average applied over a window size of 96 and a sliding window of 1. This basic unweighted moving average applies an average over a defined window size s and then moves forward in the series

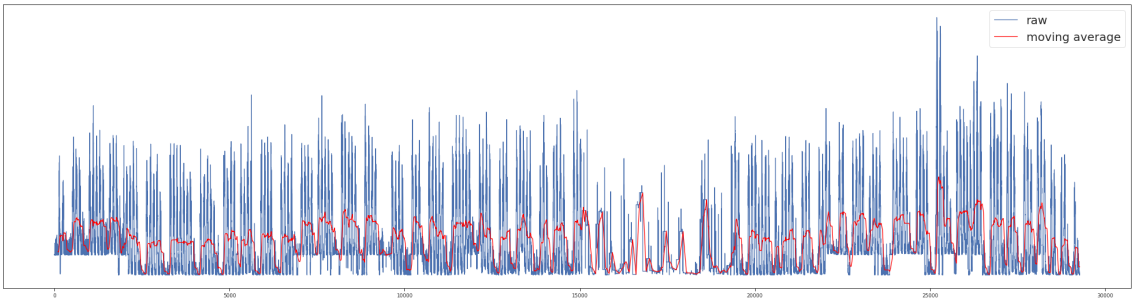


Figure 3.5: Moving average of one time series with 29 280 data points and a smoothing window size of 96

Day	*Timestamp*	*Value*	
$monday_0$	*00:00:00*	*2.56*	
$monday_1$	*00:00:00*	*2.23*	
$monday_2$	*00:00:00*	*2.45*	$\mu = 2.4$
$monday_3$	*00:00:00*	*2.37*	
$monday_4$	*00:00:00*	*2.39*	

Figure 3.6: Example calculation for moving average on same timestamp and weekday

with a sliding window step n. The resulting time series is s data points shorter than the raw time series.

Figure 3.7 shows the original time series for a weekday within a cell and the resulting smoothed time series after applying this smoothing technique. A more common behaviour with reduced noise is represented by the smoothed time series.

We don't apply a moving average in sequence over the 29 280 data points since it deletes properties of the time series that might be important. Instead, we follow the expert advice that cells in telecommunication show common behaviour among specific weekdays. This means that Mondays in sequence might be very similar to each other, but quite different to a Saturday or Sunday. Therefore, we include this knowledge in the smoothing approach. We apply a smoothing for every weekday separately and average every timestamp within the 96 data points for 5 days in sequence. Figure 3.6 can be used to understand how the smoothing is applied for Mondays on timestamp 00:00:00 with a window size of 5. This is then applied for every weekday and for every time stamp using this window size.

Monthly Averages

Another option to find a common behaviour among time series is to average the four days of the same weekday within a month. This means that all Mondays of a specific month are averaged and all other weekdays. This finds a common behaviour and reduces the amount of data noticeably. For example, a data set with 342 740 time series can be reduced to a data set with 96 178 time series. Figure 3.8 presents an example of this technique, where the red line shows the average time series and

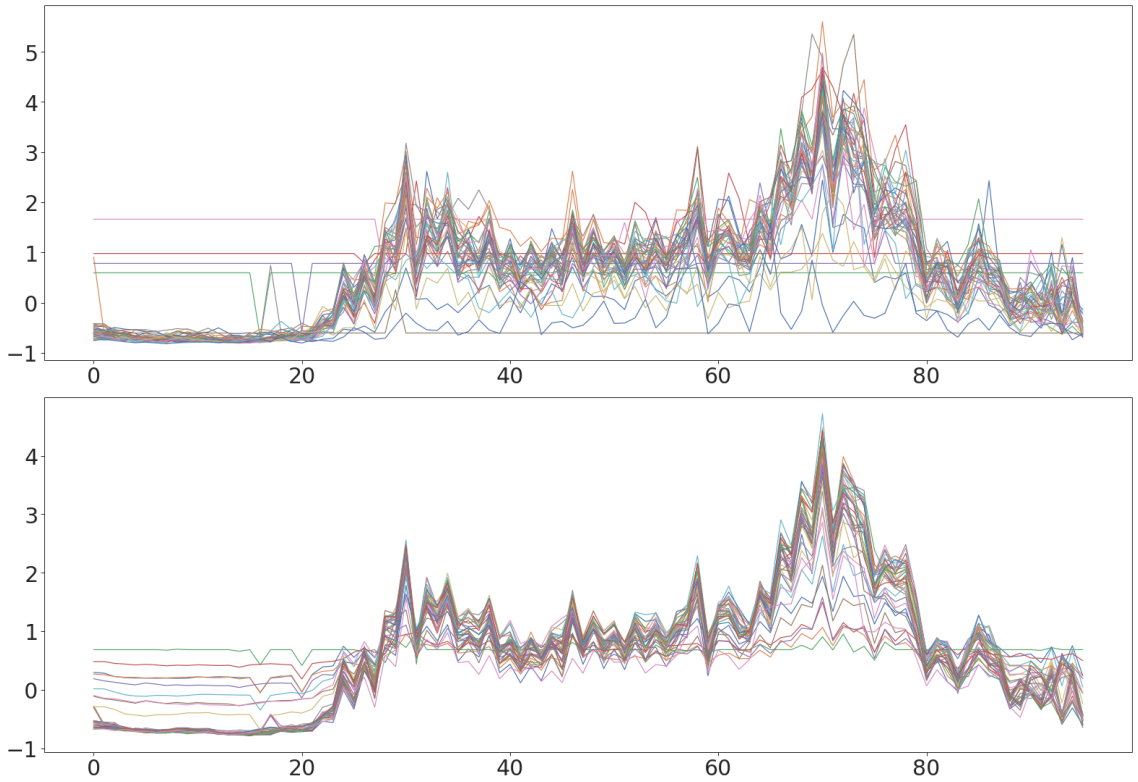


Figure 3.7: The upper figure shows the raw, normalized time series of a specific cell on a specific weekday. The lower figure shows the result of the the moving average technique. The time series show less noise and a more common behaviour.

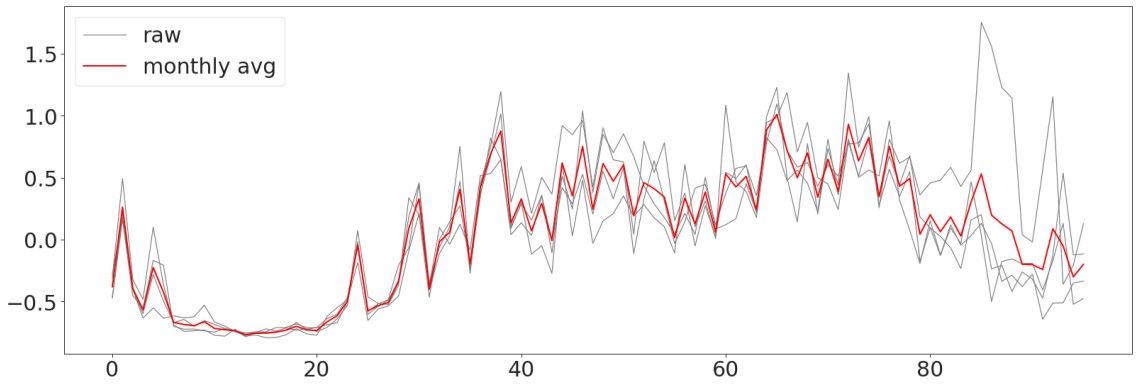


Figure 3.8: Monthly average of a specific weekday of a specific cell.

the grey lines represent the four raw time series for a specific weekday in a specific month.

3.4 Candidate Selection in Greedy Clustering

In this section, we explain how we changed the baseline algorithm referred to as Greedy Plain and described in Section 2.1.2. In this section, the general algorithm and its updates for the candidate selection are outlined. The candidate selection

Algorithm 2: Greedy Clustering (Version 2.0)

Input : data set, tau, distance measure
Output: clusterAssignments, cluster

```
# Cluster Formation:
clusters = [ ];
lookupTable = [ ];
clusterAssignments = [ ];
for each time series ts do
    features ← extract features of ts by using one of two described methods;
    if lookupTable is empty then
        append ts as new cluster in clusters;
        add new cluster in lookupTable using features
    end
    else
        candidates ← get candidates from lookupTable;
        bestCluster ← get closest cluster from candidates;
        if bestCluster < tau then
            append clusterId of bestCluster to clusterAssignments
        end
        else
            append ts as new cluster in clusters;
            add new cluster in lookupTable using features;
        end
    end
end
# Cluster assignment:
for each time series ts do
    find a cluster with minimal distance to ts using above candidate selection;
    assign ts to that clusters in clusterAssignments
end
return clusterAssignments, cluster
```

based methods are described in more detail in Chapter 3.4.1 and 3.4.2.

Algorithm 2 shows the extended Greedy Plain algorithm. The core of this algorithm is a preselection of candidates. The differences to Greedy Plain are highlighted with a yellow color. For each time series, features are extracted using either the statistical feature extraction or the transformation in the SAX representation. The `lookupTable` is empty in the first iteration, indicating that no clusters are created yet. The first time series is inserted as the first cluster and its extracted features are added to the lookup table. The structures of the lookup tables and how the feature values are inserted are described in more detail in Chapter 3.4.1 and 3.4.2. After the first iteration, cluster candidates are extracted from the lookup table by

comparing the features of the time series with the clusters and their features stored in the lookup table. These candidates are used to find the closest cluster to the time series using the ED. So instead of calculating the ED for all clusters, the algorithms with candidate selection only calculate distances to potential candidates. Here, we used a maximum of 100 candidates. The distance of the closest cluster is compared to *tau*. If the distance is smaller than *tau* the time series is assigned to that cluster. Otherwise the time series becomes a new cluster and its calculated features are added to the lookup table.

3.4.1 Greedy SAX Clustering

One approach we implement for cluster candidates selection is based on the assumption that time series can only be similar if large parts of SAX representation are shared among two time series. A similar argument is often used in approximate String matching, e.g. for genomics data. The hypothesis is that two sequences are similar if large parts (large substrings) can be found in both sequences [16, p. 474].

Therefore, we implemented a lookup table that uses extracted SAX subsequences as keys and additionally, the cluster labels where the SAX substrings are found in their centroids. For the SAX feature extraction an alphabet size of 5, a word size of 12 and a window size of 4 is used. The word size defines the target dimensionality of the time series.

Structure of lookup table

The code Listing 3.1 shows an extract of this lookup table. For each SAX substring found in a new cluster, we store the cluster indices with this specific substring in the table. At first, we stored the frequencies of the substrings in the according cluster representatives leading to a list of tuples in each key in the lookup table. The clusters did not necessarily improve by including the frequencies but the computational performance appeared to be very slow since the algorithm did an additional for-loop for each candidate selection, which increased the complexity by the number of data points. Therefore, we decided to only store the information that the SAX string occurred at least once.

```
sax_table = {'bbbb': [1, 25, 27, 30, 43, 49, 54, ...],
             'ccdd': [116, 165, 293, 371, 575, 577],
             ...}
```

Listing 3.1: Dictionary storing a list of cluster labels of cluster representative that include the SAX string

New entries in lookup table (new cluster)

Every time a new cluster is created during the Greedy Clustering, the keys in the lookup table are searched for a match to the extracted SAX strings. If the SAX string does not yet exist, a new key is created and the cluster label is stored within

Algorithm 3: Create new cluster (Greedy SAX)

Input : lookupTable, assignments, clusterRepresentatives, saxDict, ts
Output: assignments, clusterRepresentatives

clusterId ← length of *clusterRepresentatives*;
assign *clusterId* to *ts* in *assignments* array;
for each *sequence-value* pair in *saxDict* **do**
  append *clusterId* to *lookupTable[sequence]*;
end
append *ts* as centroid to *clusterRepresentatives*;
return *assignments, clusterRepresentatives*

Algorithm 4: Get candidates (Greedy SAX)

Input : lookup table, sax, cluster representatives
Output: cluster representatives, cluster labels

hits = [ ];
for *each sequence in* sax **do**
  find sequence in lookup table;
  append the cluster labels found to the hits list;
find most common cluster labels and their cluster representatives in hits;

this key. If the key already exists, the cluster label is attached to the list. The pseudo code is presented in Algorithm 3.

Get Candidates

Algorithm 4 outlines how the candidate selection using SAX substrings works. The goal is to find the closest cluster representatives and their cluster labels to the currently processed time series t based on the number of shared SAX substrings. We need to iterate over the extracted substrings for this time series and extract all the cluster labels found at the respective index in the lookup table. These labels are attached in a list hits. At the end of the loop, the cluster labels are counted and the ones with the highest count are returned together with the cluster representatives.

As a result, we reduce the number of cluster comparisons dramatically, especially when dealing with a higher number of clusters. The ED is used to find the closest cluster among all candidates returned by the algorithm.

3.4.2 Greedy SF Clustering

Another approach to select candidates is to use time series statistics to determine whether time series are similar. To extract the features, the python package `tsfresh` is used to extract 22 different characteristics from time series. This includes classical features as mean, median or maximum values or more time series specific features as number of peaks, skewness or complexity. Later in this study, an extensive feature

Algorithm 5: Create new cluster (SF version)

Input : lookupTable, assignments, clusterRepresentatives, statsDict,ts
Output: assignments, clusterRepresentatives

clusterId ← length of *clusterRepresentatives* assign *clusterId* to *ts* in *assignments* array;
for each *feature-value* pair in *statsDict* **do**
  if *lookupTable[feature]* is a list **then**
    append (*clusterId*, *value*) to lookupTable[feature];
    if *length of lookupTable[feature] > 100* **then**
      split the list into two equal lists;
  else
    bin ← find feature list with similar values to *value*;
    append (*clusterId*, *value*) to lookupTable[feature][bin];
    if *length of lookupTable[feature][bin] > 100* **then**
      split *lookupTable[feature][bin]* into two equal lists;
append ts as centroid to *clusterRepresentatives*;
return *assignments, clusterRepresentatives*

selection is outlined, where six out of the 22 features are identified as most efficient and useful to infer similarity and dissimilarity between time series.

The following aspects are explained to give an understanding how the candidate selection with statistical features works.

Structure of lookup table

It is essential to understand the structure of the lookup table to understand how clusters are created. In Listing 3.2, it is outlined how an extract of a lookup table with statistical features could look like for the feature *complexity*. The lookup table presents on the first level the feature and on the second level, the different keys that define the threshold for which cluster labels are inserted. On the third level, the cluster labels and their corresponding feature values are stored as a list of tuples.

```
stats_table = {'complexity': {
                      3.06:[(0,2.01),(2,1.98),(4,2.44)],
                      5.4: [(1,5.55)],
                      6.3: [(3,6.8)],
                      8.0: [(5,10.08),(6,12.55)],
                      18.06:[(7,19.67)],
                      22.4:[(8,25.7),(9,23.4)]}
               'n_peaks':{ ...}
               ...}
```

Listing 3.2: Dictionary storing the time series objects for every split by median of statistical feature

Algorithm 6: Get candidates (Greedy SF)

Input : lookupTable, clusterRepresentatives, statsDict
Output: candidates, mostCommonClusterIDS

```
hits = [ ];
for each feature-value pair in statsDict do
    if lookupTable[feature] is a list then
        extend hits by lookupTable[feature];
    else
        bin ← find feature list with similar values to value;
        extend hits by lookupTable[feature][bin];
hits ← get only clusterIDs from hits;
counterHits ← get counts for each counterID;
mostCommonClusterIDS ← filter for most common 100;
candidates ← filter clusterRepresentatives for mostCommonClusterIDS;
return candidates, mostCommonClusterIDS
```

New entries in lookup table (new cluster)

When a new time series leads to the creation of a new cluster, the values of its statistics are compared against the lookup table. As presented in Algorithm 5, the bin is searched as a first step, where the cluster id needs to be inserted. Therefore, the keys of a feature in the table are searched for the closest match corresponding to the statistical feature value. As long as the number of cluster labels in a key does not exceed 100, the new label is simply attached to the existing list. If the number exceeds 100, the bin needs to be split, such that 50 cluster labels are in one bin and 50 in the other bin. That means that two new keys are created, where the first is the median value and the second the maximum value. As we store the feature values in the tuples, we can sort them and split the list at the median and allocate the tuples to the two new keys. After this split, the old key with its 100 members is deleted. At the end of the clustering, the number of keys at the second level correspond to the number of bins. The splitting process ensures that the inner entries of the table are always balanced and grow dynamically with the number of clusters. This table has to be built separately for every data set due to a different distribution and scale of values.

Get Candidates

The code for selecting the candidates is presented in Algorithm 6. For every time series, this method is called to retrieve the cluster candidates before calculating the ED. Here, for every statistical feature the same procedure is followed. The core of this function is to find the accurate bin, where the statistical value of the current time series is matching. To find this bin, the sorted keys of the feature lookup table need to be iterated over to find the key in the table, where the key is greater equals the statistical feature value of the current time series. Once the correct bin is found, the cluster labels stored in this list are attached to a hit list, which stores all

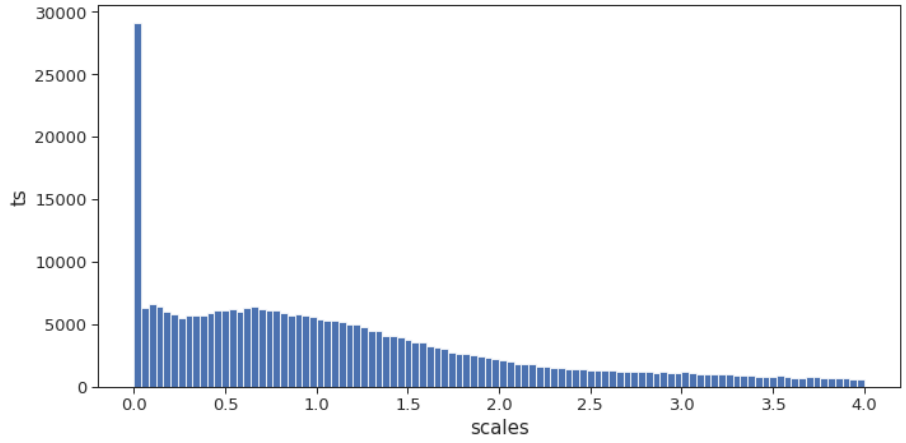


Figure 3.9: Distribution of min-max ranges of all time series in the data set.

potential candidates. The last step is to count the occurrences of cluster labels and select the 100 labels with the highest frequencies. These are the final candidates returned by the function.

3.5 Magnitude Adaptive Clustering

We started this thesis using a static *tau* in Greedy Plain. After a thorough investigation of the cluster results, we realized that the algorithm only compressed a certain set of time series very well. It produced a vast majority of single clusters compared to the overall number of clusters and few clusters with medium number of members and very few clusters with more than 80% of the entire data set grouped together. We noticed that many cluster representatives are not a good fit for their members by plotting a subset of cluster centroids with their members.

Even though, the z-normalization is normalizing the data set to a certain value range defined by the mean and standard deviation, it is still possible that individual time series in the data set show different ranges in their values. This is due to the fact that the same normalization parameters are used for the whole data set. Figure 3.9 shows the distribution of scales within the time series. We notice that many time series have a very small scale while a smaller subset has a comparably large scale. This is due to the nature of telecommunication data as the same performance measurement of different cells can show very different values. For example, two small-scale time series could have a distance of approximately 0.001 and are considered similar. Two similar moderate-scale time series could have a larger distance since the value range in the time series is larger than in small-scaled time series and be considered dissimilar by the static *tau*. Analysing the scales of the cluster members led to the conclusion that the different distributions of scales need to be considered by using a separate *tau* value based on the magnitude of the time series processed. The range value represented by $\max - \min$ seemed to be a reasonable statistical feature.

Figure 3.10 shows an abstract visualization of clusters. The numbers 1, 2, 3 and

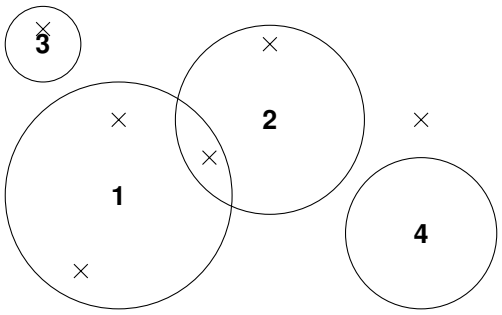


Figure 3.10: Abstract visualization of clusters with different radius. The radius of a cluster depends on the min-max range of the cluster representative. Sample data points are marked with an "×".

4 represent four sample clusters and time series are represented as an "×". The clusters have a different radius depending on their respective *tau*. A data point may be assigned to a cluster when it falls within the radius. One characteristic of Greedy Clustering is that clusters can overlap. If a time series could belong to two clusters it will be assigned to the closest centroid. If a time series falls in no radius it forms a new cluster.

Dynamic *tau* based on cluster representatives

In the altered algorithm, the value range of each cluster centroid is stored. In each iteration, the shortest distance is chosen by summing up the distances to the cluster centers with the deviation of their respective max-min-range to the max-min range of the time series that is being processed. The new distance can be formulated as follows:

$$d(ts, c) = ED(ts, c) + |(\max ts - \min ts) - (\max c - \min c)|. \tag{3.1}$$

This distance measure is only used when choosing the cluster representative with the smallest distance. For the comparison with *tau*, the ED is taken. The new *tau* value is calculated using the following equation:

$$\text{threshold} = \tau \times |\max c - \min c|, \tag{3.2}$$

where c is the closest cluster representative or the time series depending which one has the smaller min-max range. From the two range values involved in the comparison the smaller one is taken for the comparison. τ needs to be adjusted towards the desired compression. Choosing it smaller leads to more clusters and single member clusters. Increasing the value reduces the number of clusters.

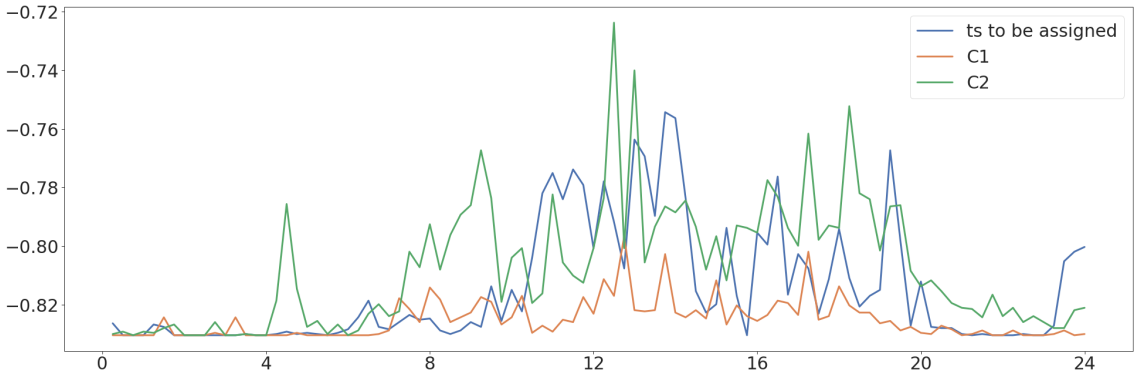


Figure 3.11: Show case of closest clusters using different distance measures. Case 1 shows the closest cluster to the blue series using the ED. In this case, the blue time series would create a new cluster since the distance exceeds the threshold for new clusters. Case 2 represents the closest cluster assigned using the distance measure proposed in Equation 3.1. In this case, no new cluster is created. From eyesight, the blue time series shows similarities with both calculated closest clusters.

	$ED(ts, c)$	$d(ts, c)$	threshold for new cluster	$\max c - \min c$
cluster, case 1	0.20769	0.25006	0.09739	0.03382
cluster, case 2	0.21072	0.24128	0.21941	0.10674

Table 3.2: Distances between the time series to be assigned (blue in Figure 3.11) with a min-max range of 0.07619 and the clusters assigned using different distance measures.

Choosing the closest cluster - a case study

Figure 3.9 shows the distribution of min-max ranges of the data set. Almost 10% from the 342 740 time series have a scale close to 0. Most of the time series have a scale smaller than 1.

By using the ED instead of the combined distance as defined in Equation 3.2, we might find a closest cluster which can be a miss match because the distance exceeds the *tau* value. This would create a new and possible single member cluster. Therefore, the second closest cluster might be a suitable fit since it is only disregarded by a very small margin. Especially for distances smaller one, the square in the equation of the ED pushes the distance down. Therefore, time series with a clearly different shape could be considered more similar on a smaller scale than on a larger scale.

The number of single clusters created by the three compression algorithms are reduced noticeably. We implemented this logic as single clusters do not compress and at the same time are not useful for sequence prediction tasks based on cluster labels.

Figure 3.11 presents three time series. The blue series needs to be assigned using a distance measure. C1 in orange is the closest cluster using the ED. C2 in green is the assigned cluster using the proposed distance measure in Equation 3.1.

Table 3.2 shows the distances between the time series and the closest cluster in both cases and the threshold for creating a new cluster and their respective ranges. Using the ED, cluster C1 is the closest. By using the ED for finding the closest cluster, we compare the distance 0.20769 with 0.09739. Here, the distance is larger than the threshold. A new cluster is created, even though we see that the time series seems to be similar to C2, which is not chosen as the closest cluster. Using the new distance measure $d(ts, c)$ (comp. Equation 3.1), we penalise a large deviation in value ranges between time series. Therefore, the distance for choosing the cluster is larger to C1 and C2 is chosen as the closest cluster. The ED between the time series and C2 is 0.21072, which is smaller than defined threshold and the time series is assigned to C2. Here we take the range of the time series to calculate the threshold since it is smaller than the range of C2.

3.6 Evaluation of Clustering

The evaluation of clustering algorithms is a challenging task due to their unsupervised nature. We extracted different information from the compressed data sets to make assumptions about whether the *tau* value used for the clustering should be adjusted. Besides this, CPU time and memory usage will be evaluated.

Descriptive statistics
A comparison of the distribution of statistical features between the original data set and various compressed data sets using diverse tau values can help to adjust tau accordingly.

Number of cluster members
The number of members in every cluster can give an indication on whether a suitable tau value is selected. Possible observations are that a few very large clusters are created and too many single member clusters if the tau value is not aligned.

Number of single clusters
The number of single clusters can represent the outliers in your data set. By using the following equation, the number of outliers can be expressed as a percentage p, where n is the number data points in the data set and s is the number of single member clusters:

$$p = \frac{s}{n} \tag{3.3}$$

Compression rate
Compression rate can be described as one minus the ratio of the compressed and uncompressed size; for instance, a data set of 100 000 time series reduced to 2000 clusters has a compression rate of 98 %. Equation 3.4 defines the

compression rate formally.

$$\text{compression rate} = 1 - \frac{n_{clusters}}{n_{dataset}} \tag{3.4}$$

The main focus of our research is improving the performance of the Greedy Clustering algorithm. To analyse the efficiency of the suggested algorithms Greedy SF and Greedy SAX, different data sets are clustered and timed to determine the differences in run time. Every clustering run is performed in the same way for the baseline Greedy Plain as for Greedy SAXs and Greedy SFs. With this practice we can investigate which clustering algorithm is more efficient depending on the amount of clusters generated.

The Greedy Clustering algorithm does not ensure a good quality by design. Since the clustering algorithm is used as a compression method, different descriptive statistics are extracted for the original time series data set and after the compression for the three different algorithms. The distributions of the results can then be compared with the original distribution to determine, which algorithm is more precise.

Additionally, we evaluated 22 statistical features in their ability to distinguish between similar and dissimilar time series and their efficiency for the candidate selection in Greedy SF. Using the Pearson's correlation coefficient, we calculated a r-value for each statistical feature that indicates if time series with a large ED are represented by a large distance in the statistical feature and vice versa. This investigation is useful to decide which features are integrated in the candidate selection in Greedy SF.

3.7 Prediction

The general setup for the prediction tasks is a train test split of 80/20 for both, the prediction based on cluster labels and the autoencoder in combination with the stacked LSTMs. In general, the data set of Counter 79 is used for the prediction task. In total, 72 cells are predicted that contain 305 days of consecutive time series. This sequence of time series is split into 244 days for training and 61 days for test. The data preparation and architectures for both models are outlined in the following sections.

3.7.1 Long Short Term Memory (clusterLSTM)

In the following, the LSTM prediction based on cluster labels is referenced as clusterLSTM. For every one of the three clustering algorithms and their respective compressed data sets, the following data engineering is applied. At first, the time series are sorted by date and the cluster labels are extracted. This resulted in three files containing the 72 cells with its 305 cluster labels, sorted by date. These labels are then transformed during the prediction into X and Y pairs. The most basic preprocessing for cluster labels is for consecutive days, where we do not consider any

different behaviour among different weekdays. Consider the following example list of cluster labels $[6, 5, 6, 46, 46, 49, 5, 46, 6, 46, 32]$.

Here, we applied a sliding window of one and a sequence length of seven to extract the labels for the test and training set. The seven previous cluster labels in X are used to predict the 8th cluster label in y. Before the training, the y labels are one-hot-encoded to be used in combination with a cross entropy loss. An extract of the data preprocessing can be seen in the following example:

$$[6, 5, 6, 46, 46, 49, 5] \rightarrow [46]$$
$$[5, 6, 46, 46, 49, 5, 46] \rightarrow [6]$$
$$[6, 46, 46, 49, 5, 46, 6] \rightarrow [32]$$
$$[46, 46, 49, 5, 46, 6, 46] \rightarrow [?]$$

Time series data as the telecommunication data used here can be processed in different ways. Two ideas on how to process the days of the cells are discussed in the following paragraphs.

Idea 1: Predicting Weekdays Separately

This is a suggestion to improve the cluster-based prediction by separating the cluster labels by weekday and predicting every weekday, separately. Due to time limitations, we could not compare the two types of label predictions. We assume that the LSTM is able to learn this behaviour. Nevertheless, we would still like to outline the idea. In telecommunication networks, recorded measures typically show a different behaviour for every weekday. This could be due to the location of the cell as it might be at an industrial park and therefore the usage is high during the day from Monday and Friday, but low on the weekend. Therefore, we wanted to split the cluster labels by every weekday and predict the next respective weekday in comparison to predicting the next consecutive day as described before. A sliding window of one and a sequence length of three is used to build the feature vectors. An extract of the data preprocessing is shown as follows:

$$[m_1, m_2, m_3] \rightarrow [m_4]$$
$$[tu_1, tu_2, tu_3] \rightarrow [tu_4]$$
$$[w_1, w_2, w_3] \rightarrow [w_4]$$
$$[th_1, th_2, th_3] \rightarrow [th_4]$$
$$[fr_1, fr_2, fr_3] \rightarrow [fr_4]$$
$$[sa_1, sa_2, sa_3] \rightarrow [sa_4]$$
$$[su_1, su_2, su_3] \rightarrow [su_4]$$

Idea 2: Similar Cells

As described in section 3.1, depending on the geographical location of cells and their surrounding population and industry level, they can show a different behaviour. On the other hand, cells in the same area might show a similar behaviour to each other. In order to increase the amount of training data and to decrease the number

Algorithm 7: Create similarity matrix

Input : dataset
Output: clusterSets, similarityMatrix

```
similarityMatrix = [ ];
clusterSets = [ ];
for each cell in dataset do
    cellSet ← get set of cluster labels;
    for each i, set in enumerate(clusterSets) do
        if cellSet and set coincide to 50 % then
            append it to the similarity matrix at index i;
            break;
    if cell not added to similarityMatrix then
        append cellSet to clusterSets;
        append cell as one-element list to similarityMatrix;
return clusterSets, similarityMatrix
```

of machine learning models to train, we can group similar cells and their cluster labels together. The prediction models are trained on a merged training set for these similar cells. The test data is evaluated individually for each cell. To identify similar cells, we needed a similarity measure. Therefore, we compared the present cluster labels in each cell. If two cells coincide by at least 50% with their cluster labels, they are considered similar. In Algorithm 7, the pseudo code for calculating the similarity is presented. The result is a matrix, where the first index or row index coincides with the according list index in the `clusterSets` variable. Each row in the `similarityMatrix` has a list of cells that are similar to each other.

Architecture of LSTM

The LSTM used for the prediction task consists of two hidden layers with 25 units and a dense layer with number of units corresponding to the number of unique cluster labels. Furthermore, we used a categorical cross entropy as loss function and Adam [12] as optimizer. The training is done for 25 epochs using a batch size of 16. In general, we tried to create a similar architecture to the benchmark prediction described in Section 3.7.2.

Moreover, we implemented a different LSTM architecture using two layers of 14 units in respect to the length of the input sequences. We wanted to investigate whether the smaller architecture produces similar results while at the same time being more efficient. The results are presented in Appendix A.3.

3.7.2 Autoencoder and Stacked LSTM (autoLSTM)

Our benchmark model against cluster cluster-based prediction consists of an autoencoder and six stacked LSTMs. In the following, this prediction is referred to

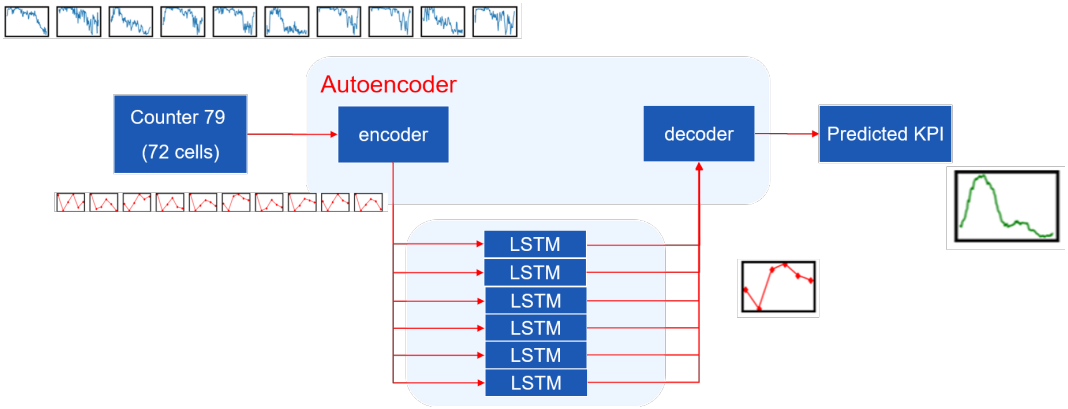


Figure 3.12: Workflow from compressing time series using encoding to reduce the data used for prediction. The prediction is applied using six stacked LSTMs predicting every future data point separately. The prediction can be decoded to the original 96 data points in the end.

as autoLSTM. The workflow is presented in Figure 3.12. The encoder compresses the 96 data points of every time series in the training data set to six data points. Examples of these compressed time series are presented in Figure 3.12 in red.

The autoencoder is trained with 20 epochs on the entire training data set of Counter 79. We used the same 72 cells for training and test as for the clusterLSTM. Once every time series is encoded by the encoder from originally 96 to 6 data points, X and y pairs are built. To achieve this, we split the 6 data points for every day and combine the values of 14 consecutive days in X and the 15h value in y. This is applied using a sliding window of one for the entire data set and for every cell. The first 80% of the pairs are used for training and the last 20% for test in the same way as the cluster-based prediction. As we have 305 days of data per cell, where 244 days are training and 61 days are test data, this results in 244 - 14 sequences for training. Then, six stacked LSTMs are used to train the data. This results in a sequential prediction of six different values for every day. The decoder mirrors the architecture of the encoder and can be used to decode the combined six values to the final prediction of 96 data points.

The architecture of every LSTM is similar and presented in Figure 3.13. Each LSTM consists of two hidden layers with 25 and a dense layer with 15 units. This architecture respects that values of 14 consecutive days are used to predict the value of the 15th day. Additionally, a L2 regulariser is applied on every layer. Moreover, we use MSE as cost function and Adam with a learning rate of 0.001 as an optimizer. The hidden layers are using the *tanh* activation function, while the output layers of the encoder and decoder use a *linear* activation function.

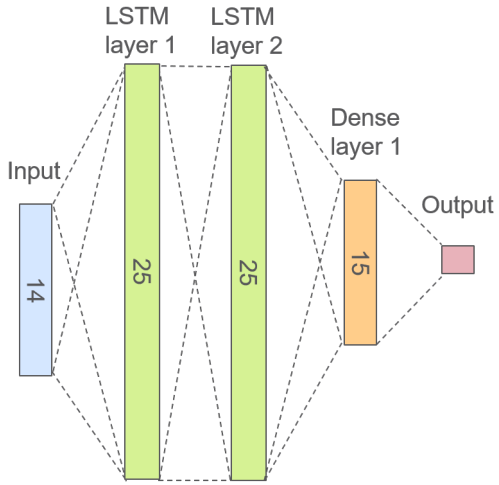


Figure 3.13: Architecture of stacked LSTM. The LSTM uses 25 units per layer and predicts the next value.

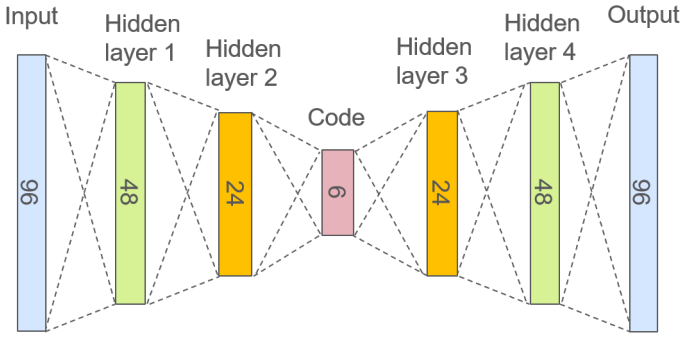


Figure 3.14: Architecture of the autoencoder. The encoder produces a code of length six from a time series of length 96. The decoder can reconstruct the time series by using the code as the input.

3.8 Anomaly Detection

Potential anomalies can be identified by using the predicted centroids and the deviation from the actual time series and the predictions. An anomaly threshold can be extracted from the predictions in the training set. We defined this threshold as outlined in Equation 3.5, where n is the number of time series in the training set and t is the individual time series. The calculation of the MSE is as outlined in Equation 2.4. The MSE is measured between the predicted centroid and the actual time series. Given that the MSE is stable for most days, the average MSE over all days of one cell should reflect an approximate of the MSE. We then define the threshold by empirically looking at MSEs on a day basis. If a MSE exceeds the average by a larger difference, it can be acknowledged as an anomaly. The following

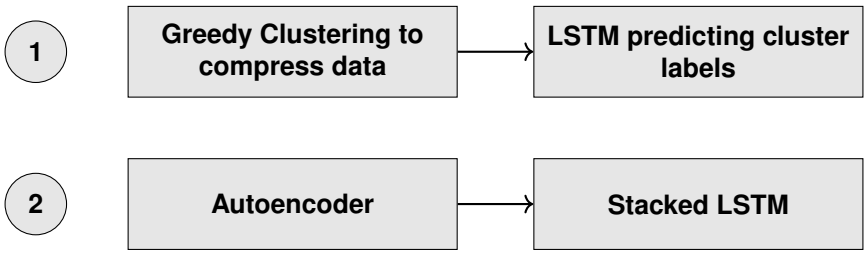


Figure 3.15: Overview of prediction pipelines. In Section 4 the first pipeline will be compared to the second pipeline.

parameter	**clusterLSTM**	**autoLSTM**
epochs	25	6 x 4 (24 total)
batch size	16	16
hidden layers	2 x LSTM (size: 25)	2 x LSTM (size: 25), 1 x Dense (size: 15)
dense output layer size	number of unique labels	1
optimizer	adam	adam
loss	categorical cross entropy	MSE

Table 3.3: Configurations of LSTM models.

threshold is defined through an empirical approach:

$$\text{threshold}_{\text{anomaly}} = 2 \times \frac{1}{\text{n}} \sum_{\text{t}=1}^{\text{n}} \text{MSE}_{\text{t}} \tag{3.5}$$

3.9 Evaluation of Machine Learning

Prediction

We selected 72 cells for Counter 79 for prediction and applied the prediction in the same way to evaluate which compression of C to more accurate cluster predictions. The cluster labels predicted by the LSTM can reconstructed to the 96 data points of the respective cluster centroid. To get more information about whether the prediction is influenced by a bad compression, we calculated the difference between the actual centroid produced by the compression and the original time series. A simple accuracy score can not be applied here, since cluster representativess can be overlapping, which can not be concluded from their cluster labels. These values can then be averaged by the number of days. Finally, we receive one averaged MSE for every one of the 72 cells for Greedy Plain, Greedy SF and Greedy SAX. The MSE scores can then be compared to a benchmark model.

Figure 3.15 shows both pipelines. The benchmark model consists of an autoencoder that is trained with the original data set. For the benchmark model we can calculate the average MSE between the predicted time series and the actual time series and

compare the results for every cell with the prediction based on cluster labels. Moreover, a comprehensive comparison of the performance between both approaches is applied including the CPU time and memory usage.

Table 3.3 shows the configurations for the clusterLSTM, which predicts cluster labels, and the autoLSTM, which uses six stacked LSTMs to predict encoded time series. The configurations are specified to be almost equivalent.

Anomaly Detection

The anomaly detection model is demonstrated by a concrete cell, where the previously described approach of using prediction errors for anomaly detection can be applied. In addition, several cells have been analysed in regard to this approach.

4

Results and Discussion

The results are introduced by a brief exploration of the cluster results for Greedy Plain to deepen the understanding of the clusters and their members, differences between cells and weekdays of one cell. The exploration is followed by selecting six statistical features out of 22 for the candidate selection in Greedy SF and a general comparison of efficiency between the SAX substrings and the statistical features. Furthermore, a performance comparison of the algorithms Greedy Plain, Greedy SF and Greedy SAX is presented. The clustering results are concluded by an extraction of descriptive statistics for the original time series and the three compression algorithms to evaluate the change in distribution. Furthermore, the machine learning results for the LSTM prediction on the three compressed data sets for 72 different cells are presented. This section is followed by the results of the benchmark model autoLSTM. The results are compared and discussed including a comprehensive performance evaluation of the two different solutions for prediction. Finally, a proposal for a method to identify anomalies based on the clusterLSTM is presented including a representative example.

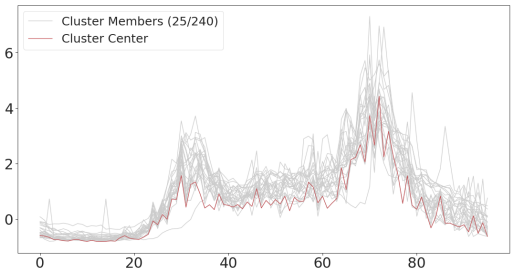
4.1 Exploration of Cluster Results

Figure 4.1 presents four cluster with its 25 first members. The number of members in total are presented in the legend. We used these visualizations to get a feeling for the clusters created by the compression and for tuning the *tau* value. We adjusted the *tau* value such that more larger clusters with several hundred candidates and different shapes on different scales were created.

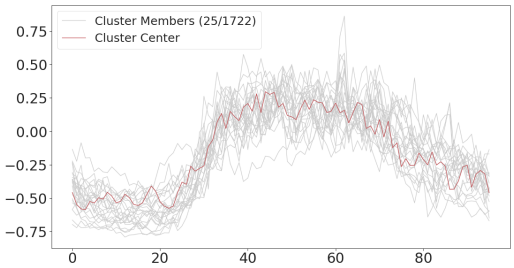
Moreover, we stored the set length of the cluster labels present of every cell in a list and visualized the results in a histogram in Figure 4.2 for Counter 79. Given that the compression and the respective cluster representatives for every day and every cell is close to the original time series, this visualization can be used to identify more constant and less constant cells. We assume that more stable cells would lead to less variation in cluster labels and vice versa.

An important insight we have received from domain experts is that the data looks quite distinct from weekday to weekend. Therefore, we visualized the occurrence of cluster labels and the respective counts distinguished by weekday in blue and

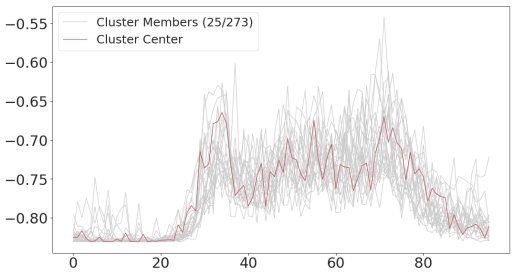
(a) Cluster 1



(b) Cluster 23



(c) Cluster 32



(d) Cluster 53

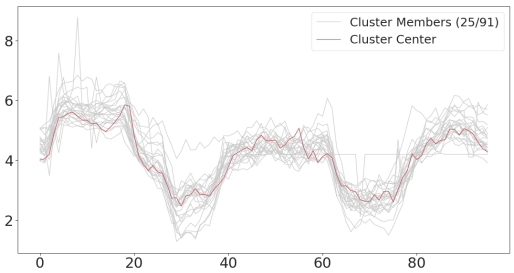


Figure 4.1: Figure (a) - (d) show 4 exemplary clusters with different shapes from the Greedy Plain compression for Counter 79.

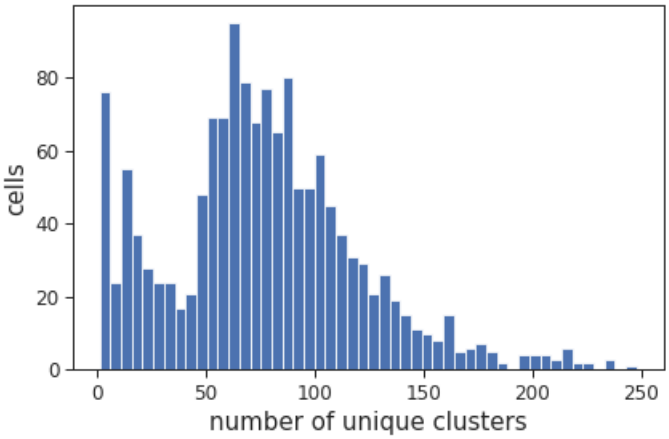


Figure 4.2: Histogram presenting the number of unique clusters for every cell in Counter 79.

weekend in orange in Figure 4.3. A clear distinction between work days and weekends can be observed.

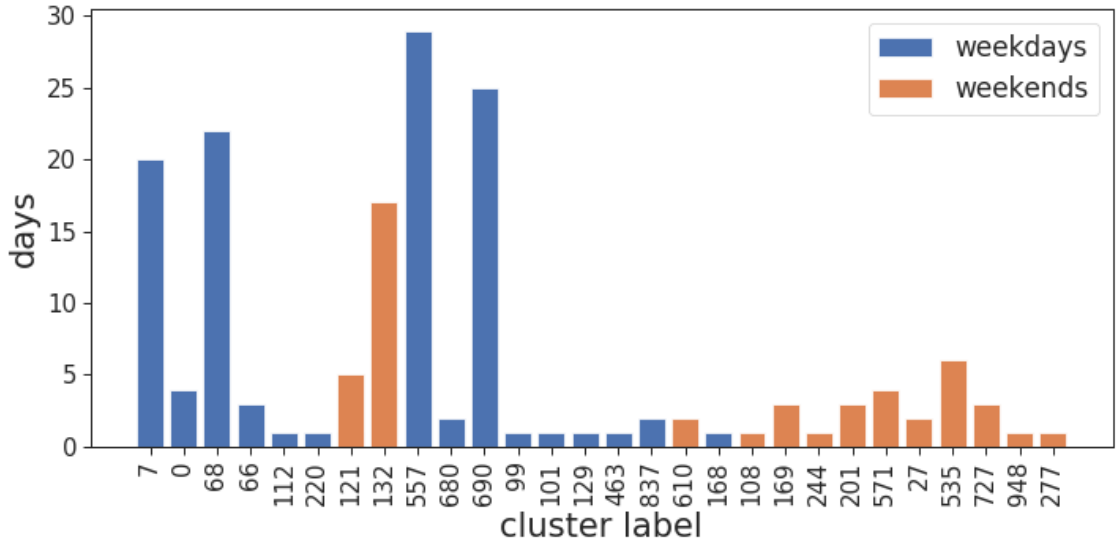


Figure 4.3: Frequency of cluster labels for one cell. The cluster labels present during the week from Monday to Friday are marked in blue, whereas the weekend is marked in orange.

4.2 Clustering

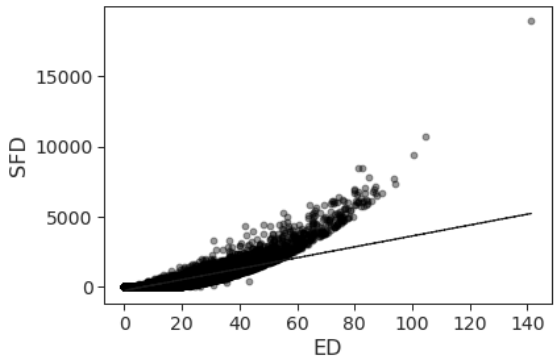
4.2.1 Feature Selection in Greedy SF

Various statistical features can be extracted from a time series. The computation of the features can be expensive for large data sets as they need to be computed for every time series. Therefore, the features should be selected first by their computational complexity and the quality for inferring the similarity or dissimilarity between time series.

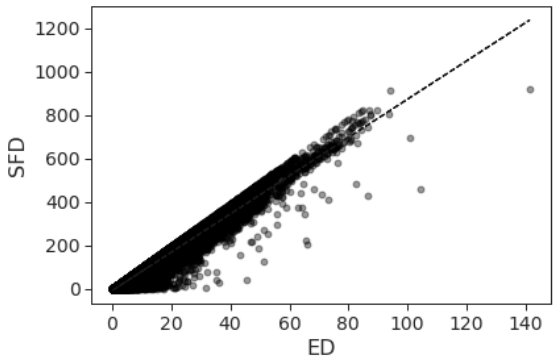
To analyse how useful a statistical feature might be to distinguish between similar and dissimilar time series, we investigated the distribution of the values for 22 different statistical features. Only continuous features are included for this investigation, whereas discrete features, like number of peaks and Boolean values, are excluded. The distribution of features is a first indicator for their applicability. For the measurement of the distribution, a histogram was computed for every list of features using 100 bins. Here, it was measured how frequent a bin contained at least 1% of the time series included in the data set. Furthermore, we measured the necessary time to compute the features.

To examine the quality of a statistical feature for the candidate selection, the distribution alone is not sufficient. Hence, we investigated, whether a relation of the ED between two time series and the absolute difference between their statistical features exists. To achieve this, we analysed for a given pair of time series t_1 and t_2, whether the $ED(t_1, t_2)$ is large if the difference between statistical feature values $SFD(t_1, t_2)$ is large and vice versa. Therefore, we sampled 50 000 pairs of time series randomly with replacement and computed the ED and the absolute differences of all statistical

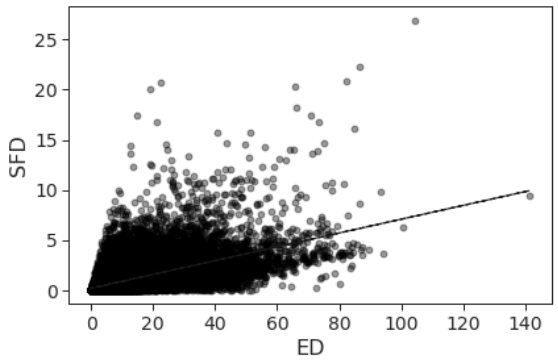
(a) Absolute energy (R = 0.83)



(b) Sum values (R = 0.98)



(c) Complexity (R = 0.63)



(d) Mean (R = 0.98)

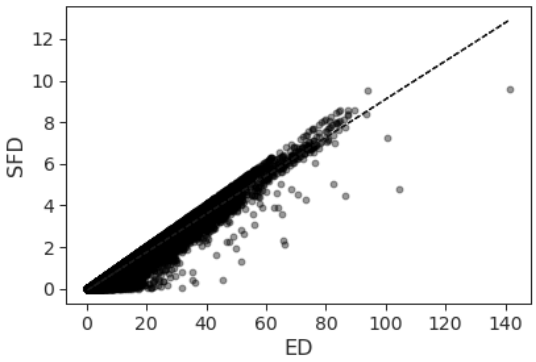


Figure 4.4: Distribution of statistical feature values for Counter 79.

features for every pair. These values can be plotted and the R-value computed. The closer the R-value is to 1 and the clearer the trend is visible, the better the feature is suited to be used in the candidate selection. The results of this investigation are presented in Figure 4.4. It presents the results of four different statistical features.

The results for the feature absolute energy are not entirely obvious. There is a visible relationship and the correlation coefficient is rather high with a value of 0.815. Nevertheless, it does not show a clear linear relationship but a skewed one. Many pairs between a ED of 0 and 20 show very similar values for the feature. We did not use this feature for the candidate selection, but it could be used based on this investigation. In general, it is advisable to analyse the results in detail and to not rely completely on the correlation coefficient R. Sum of values has a R value of 0.933 and shows a clear linear dependency and a high distribution of statistical values. We measured a high R value of 0.976 for the mean value and included this feature. Complexity is a good example for a feature that is not useful as no linear dependency is observable. The comprehensive results of this feature selection are summarized in Table 4.1. We highlighted every feature with a reasonable efficiency, variance and correlation using a green colour. These results only apply to Counter 79. It must be noted that the results might deviate for other data sets. Moreover, we suggest using more statistical features for larger data sets as they will lead to

statistical feature	time in seconds	n_bins >1% ts	Pearson correlation coefficient
First Quantile	412	7	0.771
Third Quantile	419	13	0.919
absolute energy	3.09	2	0.815
absolute sum of changes	12.14	23	0.726
complexity	11.55	16	0.647
kurtosis	156	18	-0.056
maximum	6.38	16	0.718
first maximum	3	53	0.024
minimum	5.5	4	0.387
no peaks (distinct values)	86.5	excl.	excl.
count below mean	15.1	32	-0.049
mean change	1.4	6	0.394
standard deviation	25.7	13	0.788
median	43	10	0.898
mean	45	9	0.976
mean second derivative	1.8	7	0.328
sum values	4.94	10	0.933
skewness	147	23	-0.008
count above mean	12.87	32	0.0573
first minimum	3	29	-0.064
longest strike above mean	107	25	0.118
longest strike below mean	110	30	-0.068

Table 4.1: Computing time for 15 different statistical features and a count how many times at least 1% of the data was present in a bin. The histogram used for this calculation was created with 100 bins and for Counter 79 including 342 740 time series. The selected features meeting the performance and quality expectations are highlighted.

more clusters in general and therefore increase the probability that close candidates will be selected. Some features needed extensively more time to compute as for instance, the statistical quantiles. It required more than 400 seconds to compute these quantiles in comparison to the standard features like maximum, mean and medium that only required 5 - 10 seconds. Based on the degree of variance in the individual time series, features as mean could become uninformative. In this case, statistical features could be extracted on subsequences, e.g. on every 24 of the 96 data points. Therefore, it is important to carefully evaluate the usefulness and the performance for every feature.

time statistics for every counter	79	47
n time series	342 740	411 000
stats (6 selected features)	92	104
sax (word_size=24, alphabet=5, word_split=4)	321	384
sax (word_size=24, alphabet=4, word_split=4)	320	386
sax (word_size=12, alphabet=4, word_split=4)	309	377

Table 4.2: Time measurements of feature extraction for Greedy SF and Greedy SAX

total clusters	single clusters	clustering time	alphabet	window
9 291	3 905	2 194	4	24
6 057	3 807	1 095	4	12
6 147	3 878	916	5	12

Table 4.3: Different SAX parameters and their influence on the clustering results. This evaluation is done using Counter 79.

4.2.2 Clustering Performance

Different parameters

The performance of the candidate selection in Greedy SF and Greedy SAX is partially dependant on the efficiency of the feature extraction. Table 4.1 contains the computing time for the six statistical features selected in the previous Chapter. Based on the evaluation and the performance, we decided to only include absolute sum of changes, maximum, standard deviation, median, mean and sum of values in the final candidate selection.

To compare the computing time of the SAX extraction with the statistical feature extraction, we simulated the feature extractions for Counter 47 and 79. The results are presented in Table 4.2. These findings explain partially, why the performance of Greedy SAX is slightly worse than Greedy SF. Whereas the most efficient configuration of sax parameters needed 321 seconds to compute for Counter 79, it only required 92 seconds to compute the selected statistical features.

Moreover, we evaluated how different SAX parameters change the compression. The results are presented in Table 4.3. An important finding is the strong influence of the settings on the number of clusters created. Whereas the number of single clusters is almost the same for the three different SAX configurations, the number of total clusters varies noticeably between using a window size of 24 or 12. Based on the results, we decided to use the latter configuration for the predictions.

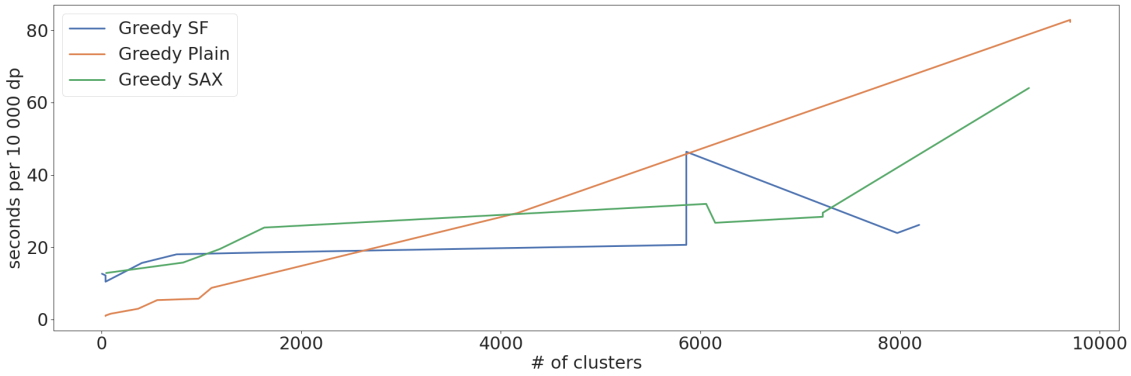


Figure 4.5: Comparison of clustering time for different versions of Greedy Clustering.

Clustering Times

Figure 4.5 outlines the time in seconds required to compress 10 000 time series depending on the number of clusters created during the clustering. For this analysis the data sets of Counter 47 and 79 were used, whose details can be looked up in Table 3.1. Every evaluated clustering algorithm uses the magnitude adaptive clustering with a dynamic τ described in Section 3.5. The development environment is shared between different users. Therefore, we only monitored the clustering times at night or on the weekend to ensure similar preconditions. Both candidate selections Greedy SF and Greedy SAX clearly outperform Greedy Plain when the number of clusters increases. For a small number of clusters, Greedy Plain is faster than the Clustering algorithms using a candidate selection. Greedy SF shows a better performance for more than approximately 2 300 clusters. Greedy SAX on the other hand, shows a better performance for more than 4 000 resulting clusters. In general, we could observe that a larger data set leads to more clusters. Appendix A.3 presents the clustering run times per 10 000 data points for a higher number of generated clusters. Greedy SF clearly outperforms the other Greedy Clustering algorithms.

Table 4.4 shows the clustering results for Counter 79 of two data sets. Generally, Greedy SF and Greedy SAX generate more clusters for the same *tau* than Greedy Plain. This is due to the possibility that the candidate selection sometimes misses potential clusters that are within the radius. In relation to the numbers of clusters generated, Greedy Plain is always slower than the clustering methods with candidate selection. We included a data set that has more than one million time series to see how the clustering performs for larger data sets. With the same *tau* of 2.88, Greedy SAX generate approximately 20 000 clusters more than Greedy SF. In this case, Greedy SAX needs considerably more time for the clustering. Greedy Plain needs over 13 hours to generate 27 555 clusters from the large data set, while Greedy SF needs less than two hours two generate over 80 000 clusters from the same data set. In addition, Greedy SAX needs noticeably less time for 19 664 clusters than the Greedy Plain algorithm for their respective *tau*. As the number of generated clusters grows, Greedy SAX shows a decreasing computational efficiency.

data set	tau	method	clusters	single clusters	clustering time (sec)	compression rate (%)
342 740	1.92	Greedy SF	54 435	31 601	1263	84.12
		Greedy Plain	27 993	15 467	8 472	91.83
	2.40	Greedy SF	16 295	6 186	885	95.25
		Greedy Plain	9 704	3 484	2 840	97.17
		Greedy SAX	49 488	46 416	3 693	85.56
	2.88	Greedy Plain	4 168	856	1 009	98.78
		Greedy SF	5 858	1 221	707	98.29
	3.84	Greedy SAX	6 147	2 269	916	98.21
1 016 489	2.40	Greedy Plain	27 555	9 789	49 256	97.29
	2.88	Greedy SF	86 717	42 787	5 115	91.47
	2.88	Greedy SAX	104 123	102 841	45 404	89.76
	3.84	Greedy SAX	19 664	19 058	6 934	98.07

Table 4.4: Clustering results for Counter 79 for two data sets with different sized data sets. The first column shows the number of time series in the data set. The three clustering methods were tested with different tau values. The clustering time is measured in seconds. All clustering results use the magnitude adaptive clustering and dynamic tau. Clustering results using the static tau are not included.

In conclusion, the statistical-based candidate selection is much more efficient than the SAX-based candidate selection, especially for data sets larger than one million time series. This could be caused by the difference in the lookup structures for the candidate selections. The lookup table used for Greedy SF is built in a way that the bins have a limited size of 100. Therefore, the hit list from which the potential candidates are chosen is limited to 100 times the number of statistical features used in the extraction. Greedy SAX has no limitation on the length of the entries in the lookup table. Thus, the hit list grows very large from which the potential candidates are chosen. This makes the computational operation in Greedy SAX more expensive as the number of clusters increases. Greedy Plain shows no competitive results and is therefore not recommended to be used for data sets larger than one million time series.

Static *tau* vs. Dynamic *tau*

It required a thorough and deep data exploration phase to understand, why the clustering using a static *tau* value for every time series in the different data sets did not result in a satisfying compression. We realized that the Greedy Clustering is highly influenced by different scales of time series. Due to a static *tau* value, time series with a small scale are grouped together in large clusters even though the relative difference within the cluster is very high. The opposite applies for time series with a large scale of values. These are often translated to single clusters or

small clusters as small differences in variation with a large scale are punished by the ED and therefore, no clusters with a smaller *tau* value are found. Our gained awareness for different scales within different cell types that could not be handled by the initial z-normalization before the compression, resulted in a breakthrough. The implementation of a dynamic *tau* calculation within the clustering relative to the scale of the time series processed resulted in much better clusters.

Due to these improvements, we decided to exclude the evaluation of the compression on smoothed data sets

Data Reduction

We stored the time series with meta data as outlined in Section 3.1. Hence, our data files are considerably larger than necessary. A data set for Counter 79 of the filtered subnetwork consisting of 342 740 time series is 943 MB large without any compression. We can reduce the stored data by only storing the most important information:

- cell name
- date
- Counter data z-normalized
- assigned cluster label
- Counter name

The file with reduced information only has 345 MB. By compressing the time series by clustering, we created a compressed file for the time series prediction and a model file that stores every information about the model learned including the cluster representatives. If we keep the important meta data in the resulting data set file it is 75 MB large. A reduced representation could reduce it to 11 MB. The resulting model file is 13 MB large. We compressed 345 MB to 75MB + 13MB = 88MB, which is 25% of the input and therefore presents a noticeable advantage regarding the data storage.

Figure 4.5 can give a good estimate on when to use the candidate selections or Greedy Plain. Considering that the tested data sets are a subset with approximately 900 MB of a data set that represents 12 GB of data, we can conclude that a candidate selection is highly necessary. Nevertheless, this is a lossy compression, which is not applicable for every use case. Therefore, it needs to be evaluated whether this type of compression is adequate for other applications. In our case, the reduction of the data set might indicate that algorithms trained on this data could be accelerated.

4.2.3 Inferring Descriptive Statistics

We investigated, whether comparable cluster representatives are created for Greedy Plain, Greedy SAX and Greedy SF. The purpose of this analysis is to evaluate how much the distribution of statistical features changes after the compression. Ideally, the distribution would be very close to the distribution of the original data set. We extracted every feature for every cluster representative of the model. Then, we counted the number of members for every cluster and propagated the statistical

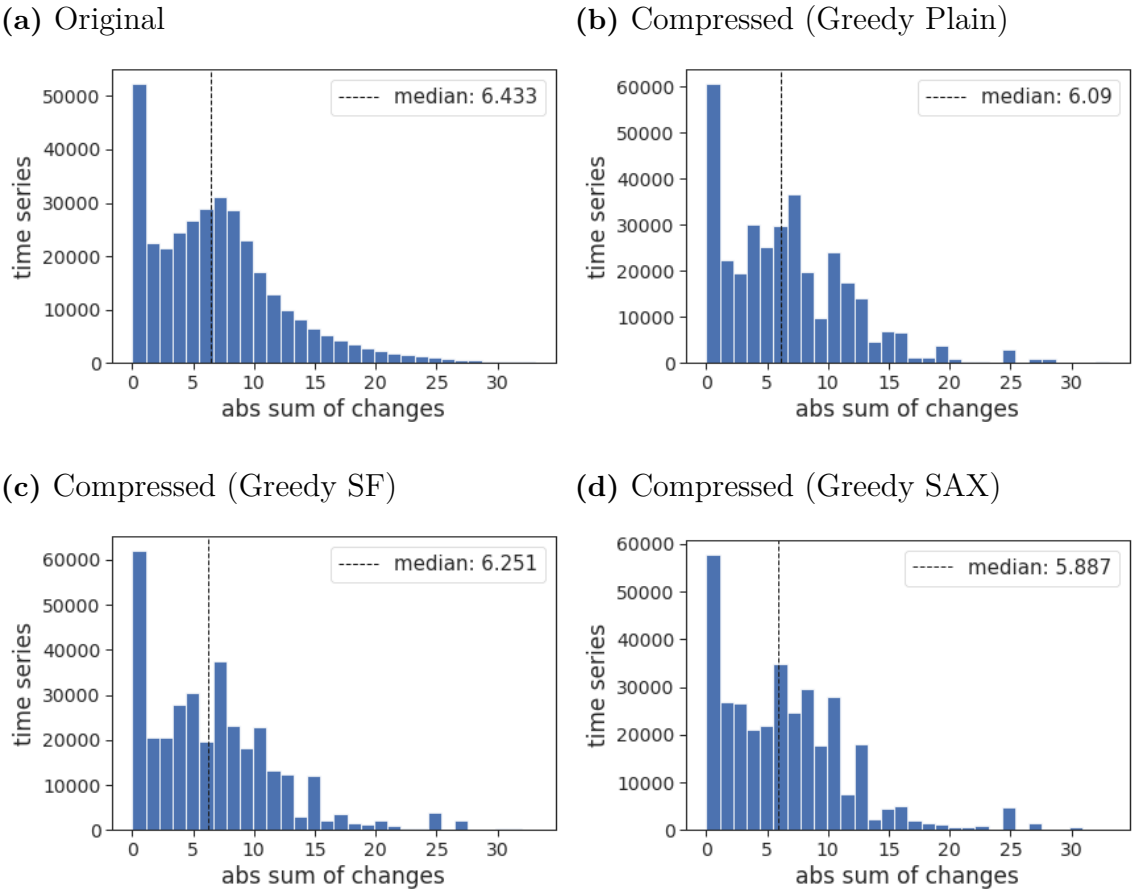


Figure 4.6: Comparison of the absolute sum of changes before and after the compression for the three clustering methods Greedy Plain, Greedy SF and Greedy SAX.

	median original	median compressed		
feature	**Original**	**Greedy Plain**	**Greedy SF**	**Greedy SAX**
abs sum of changes	6,433	6,090	6,251	5,887
maximum	0,379	0,302	0,303	0,296
median	-0,261	-0,264	-0,262	-0,307
mean	-0,230	-0,244	-0,235	-0,230
std	0,249	0,231	0,230	0,225
sum values	-22,048	-23,416	-22,597	-22,077

Table 4.5: Comparison of six descriptive statistical features for the three clustering methods.

feature value of the cluster representative by the number of cluster members. After this transformation, the distributions can be compared as they have equal length corresponding to the number of time series in the data set. We selected the six features used within the candidate selection for this investigation and used the median value of the distribution for comparison.

Counter	clustering method	tau method	tau	clusters	single clusters
79	Greedy Plain	static	1.92	60 080	47 867
	Greedy Plain	dynamic	2.4	9 704	3 484
	Greedy SF		2.88	5 858	1 221
	Greedy SAX		3.84	6 147	3 878

Table 4.6: Compressed data sets used for prediction. The original data set holds 342 740 time series.

The distributions of the statistical feature *absolute sum of changes* are visualized in Figure 4.6 and the overall results for the features are presented in Table 4.5. In Figure 4.6 slight changes in distribution can be observed, whereas the median value is close to the original in every compressed method. Nevertheless, there are differences in detail as Greedy SF and Greedy Plain show a more similar distribution and median value as Greedy SAX. This detailed observation applies to the five other features. In general, Greedy SAX shows the highest deviations to the original median value for every feature, whereas Greedy SF shows the smallest deviation.

4.3 Prediction

The methods and detailed architectures of the machine learning models used for the prediction tasks are described in Chapter 3. For the following prediction results Counter 79 was used, compressed by the three clustering methods using a dynamic tau. The old clustering version using a static *tau* was included for Greedy Plain, to analyse the difference in cluster quality and therefore the impact on the machine learning tasks. More detailed information about the compressed data sets can be found in Table 4.6.

4.3.1 Clustering and LSTM Prediction (clusterLSTM)

Table 4.7 presents an extract of the prediction results for 11 different cells. The complete results of the clusterLSTM prediction for 72 cells can be found in Appendix A.1. On the left hand side, the results for Greedy Plain using a static *tau* in comparison to the improved version using a dynamic *tau* are presented. The results indicate that a dynamic *tau* improved the fit of the clusters dramatically. As a result, the improved clusters show decreased prediction errors.

When comparing the prediction errors between Greedy Plain, Greedy SF and Greedy SAX, it can be concluded that the errors are very close among the three models for most cells. To get a deeper understanding of the influence of the compression on the prediction results, the compression with the smallest error is included in Table 4.7. Greedy SF and Greedy Plain showed the lowest MSE 27 times, whereas Greedy SAX had a frequency of 18. Greedy Plain and Greedy SF have an almost similar MSE between the true time series and the assigned centroid. Greedy SAX shows some noticeable differences and therefore leads to a slightly worse prediction.

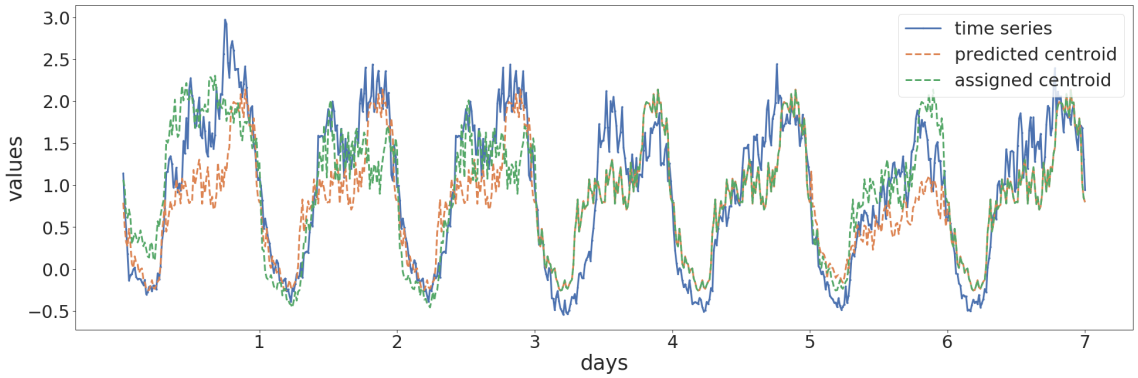


Figure 4.7: Comparison of predicted and actual cluster centroids to the original time series of cell AGB for one week using Greedy SF compression.

	Static Tau		Dynamic Tau						
	Greedy Plain		**Greedy Plain**		**Greedy SF**		**Greedy SAX**		
cell	pred	cluster	pred	cluster	pred	cluster	pred	cluster	best
AAC	4.70	1.76	0.473	0.164	0.448	0.157	0.637	0.252	stats
AAJ	2.47	1.38	1.375	0.100	1.443	0.101	1.923	0.334	plain
ABY	2.21	1.65	0.260	0.163	0.362	0.191	0.342	0.284	plain
ACC	6.22	1.15	0.594	0.126	0.663	0.148	0.433	0.132	stats
ACF	1.16	0.60	0.228	0.097	0.275	0.126	0.317	0.142	plain
ACN	4.10	0.96	0.705	0.157	0.636	0.198	0.755	0.261	stats
ACP	3.52	0.40	0.146	0.064	0.134	0.066	0.608	0.093	stats
ACU	1.90	2.10	1.026	0.148	0.937	0.193	1.201	0.210	stats
ADQ	2.65	1.68	0.433	0.254	0.402	0.251	0.612	0.287	stats
AGB	2.65	1.11	0.400	0.155	0.519	0.191	0.352	0.219	sax
ASB	6.27	3.76	4.823	0.290	5.700	0.285	5.117	2.494	plain

Table 4.7: Average MSE between predicted centroids and true time series in the test for Greedy Plain using a static and dynamic *tau*. Additionally, the error of the assigned centroids is presented to validate the compression. Furthermore, the prediction results of Greedy SF and Greedy SAX using a dynamic *tau* are presented.

Figure 4.7 presents an example of the clusterLSTM prediction of one week within one cell of the test set using Greedy SF. Moreover, we visualized the assigned centroids to see the deviation between the true time series and the assigned centroids and the predicted centroids.

4.3.2 Stacked LSTM Prediction (autoLSTM)

We compared our pipeline with an autoencoder that reduces the dimensionality of the time series to six values and applies a stacked LSTM. The predicted time series can then be decoded again to result in a prediction of 96 data points. For the analysis, we investigated the quality of the results. The CPU time and memory usage of both pipelines are discussed in the next section. All LSTMs were trained using four epochs and 80% of the data.

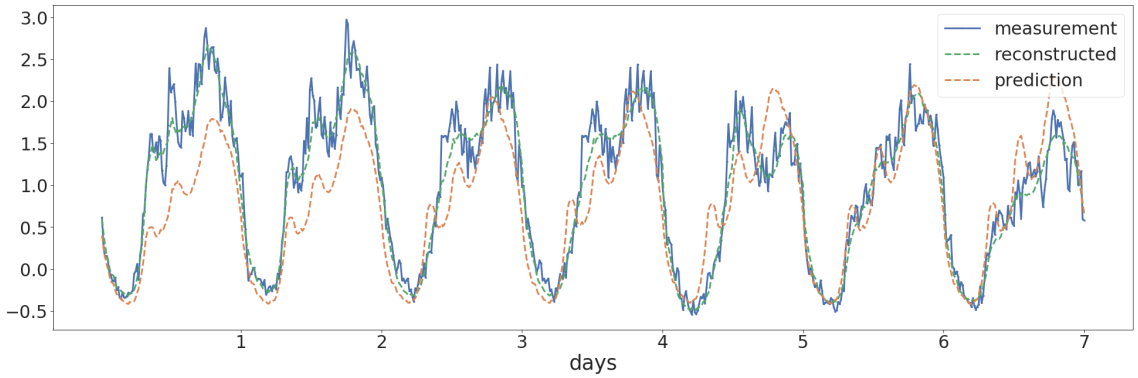


Figure 4.8: An example result of a prediction using the autoLSTM for the cell AGB. The blue line shows the raw time series for the first predicted week. The green line shows the reconstructed time series. The orange line represents the reconstructed predicted time series.

Figure 4.8 shows an example prediction of a week for the cell AGB. The reconstructed time series represents the raw time series very well and the predictions are precise despite some larger deviations on day one and two. Table 4.8 shows an excerpt of the prediction results using the autoLSTM and Figure 4.9 visualises the prediction errors. A comparison of the results using the clusterLSTM for all 72 cells is presented in Appendix A.2 and the visualisation can be found in Appendix A.1 and A.2. Most of the reconstructed time series using the decoder have a very small MSE for all cells. Nevertheless, the predictions on the encoded time series have a larger error in comparison to the reconstructed time series. Sometimes the error is very large as in cell ASB which could point to an anomaly. Figure 4.10 shows the outbreak of the time series on the second day. This is most likely an anomaly that was learned by the autoencoder. Since the prediction is mostly predicting a pattern it could not have predicted this peak. This shows that the prediction can be used as an anomaly detection.

Comparing these findings with the results of our clustering approach, it can be derived that the assigned clusters generally have a higher error then the reconstructed time series. We assume that this is one explanation for the clusterLSTM showing a slightly higher MSE than the results of the autoLSTM. Nevertheless, the errors are comparable, proving our assumption that our clustering approach presents a competitive option.

4.3.3 Comparing Resources

Besides the errors of the compared models, we measured the CPU time and memory consumption for both scenarios. To measure the CPU time, we used the Python module *line_profiler*. This tool evaluates the CPU times of a function decorated with `@profile`. As a result, it prints the overall times, times per hit and the relative amount of time spend line by line. To evaluate the memory usage, we used the tool *memory_profiler*. It works similar to the *line_profiler* and prints the overall memory and the memory increment for all profiled functions. Listing A.1 shows an example

cell	MSE true timeseries			
	autoLSTM		Greedy SF + clusterLSTM	
	prediction	reconstruction	prediction	cluster
AAC	0.163	0.018	0.448	0.157
AAJ	0.121	0.023	1.443	0.101
ABY	0.198	0.019	0.362	0.191
ACC	0.249	0.032	0.663	0.148
ACF	0.098	0.011	0.275	0.126
ACN	0.556	0.019	0.636	0.198
ACP	0.091	0.009	0.134	0.066
ACU	0.333	0.035	0.937	0.193
ADQ	0.188	0.033	0.402	0.251
AGB	0.348	0.040	0.519	0.191
ASB	29.952	0.812	5.700	0.285

Table 4.8: Excerpt from the autoLSTM prediction in comparison to the clusterLSTM prediction using compressed data provided by Greedy Plain. For the clusterLSTM, MSEs are computed between the prediction and the true time series or the assigned cluster. For the autoLSTM, MSEs are computed between the reconstructed time series and the true time series.

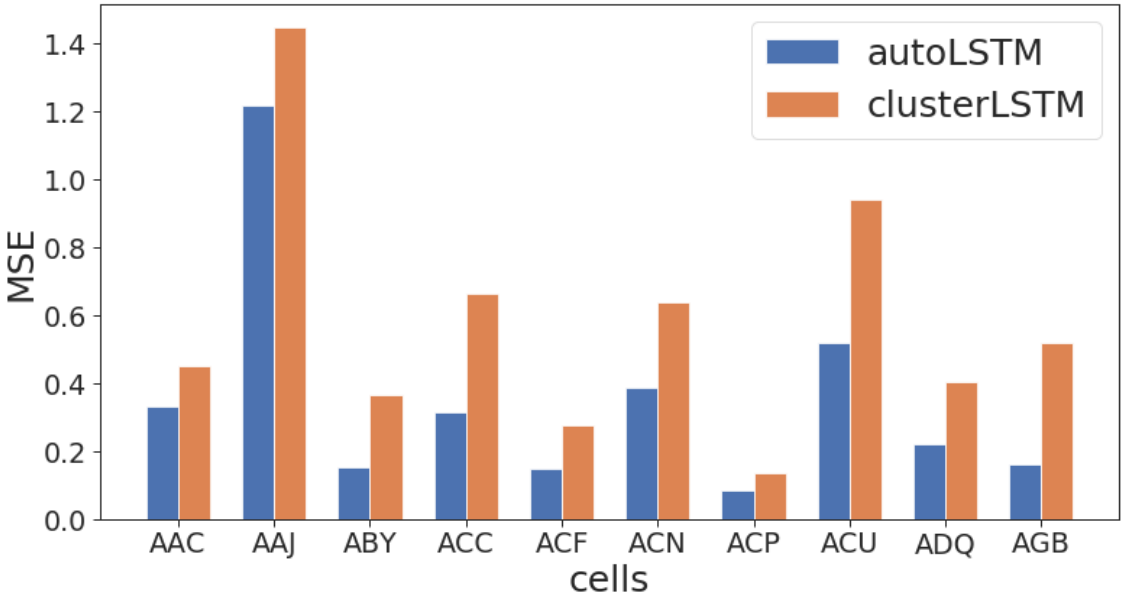


Figure 4.9: Comparison of prediction MSEs between clusterLSTM using Greedy SF clustering and autoLSTM for the selected cells presented in Table 4.8.

output of the *line_profiler*.

Table 4.9 presents the CPU time and memory usage for the three clustering methods as well as the autoencoder. The autoencoder requires less memory and time for compression than all clustering methods, but Greedy SF shows a competitive run

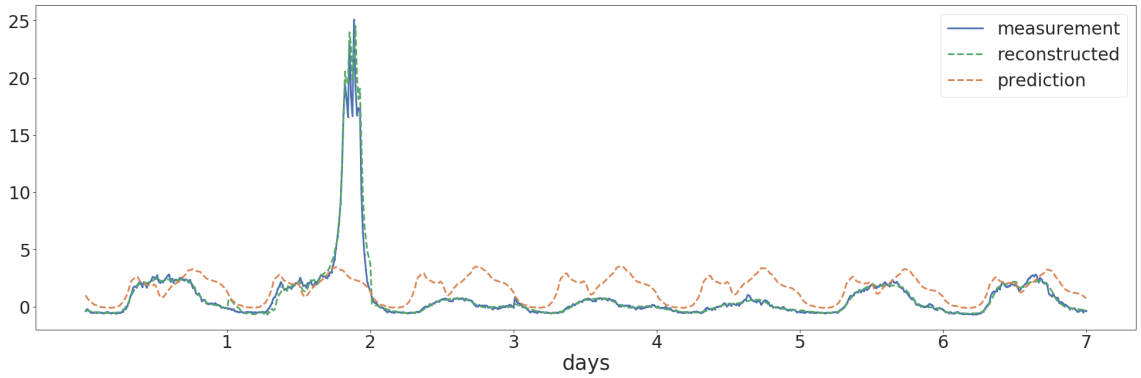


Figure 4.10: autoLSTM prediction for cell ASB. The blue line shows the raw time series for the first predicted week. The green line shows the reconstructed time series. The orange line represents the reconstructed predicted time series.

	Clustering			**Autoencoder**
	Greedy Plain	Greedy SF	Greedy SAX	
CPU Time	2 820.3	707.0	916.0	483.0
Memory usage	5 406.7	5 400.0	5 422.0	1 523.0

Table 4.9: Comparison of CPU time and memory consumption between Greedy Clustering and the autoencoder. The CPU time is measured in seconds and the memory usage in MB. The autoencoder is trained for 20 epochs.

	clusterLSTM				**autoLSTM**		
	train	evaluate	predict	total	encoder	fit/predict	total
CPU Time	804.5	128.6	109.0	1068.9	6.3	1910.5	1916.8
Memory	2263.5	7.3	12.5	3720.5	166.1	67.6	3620.7

Table 4.10: Comparison of CPU time and memory consumption for both prediction approaches: clusterLSTM and autoLSTM. Both models were trained on 72 cells including 230 days of training to predict 75 days of test. The CPU time is given in seconds and the memory usage in MB.

time. The autoencoder is trained using 20 epochs. We tested different number of epochs and according to the MSE between the reconstructed and the original time series, 20 epochs was the best balance between efficiency and quality. By modifying this parameter or implementing an early stopping mechanism, the CPU time could be further reduced. However, for the following prediction task, the clusterLSTM is almost twice as fast per cell. This could be due to the architecture of the autoLSTM as it consists of six LSTMs and therefore needs more resources in time and comparable resources in memory. The clusterLSTM prediction required 14.8 seconds to train and predict one cell, while the autoLSTM using edcoded time series needed 26.6 seconds. The total time of predicting 72 cells using Greedy SF and the clusterLSTM required 1 775.9 seconds, whereas the autoLSTM needed 2 398.2 seconds. This is also visualized in Figure 4.11.

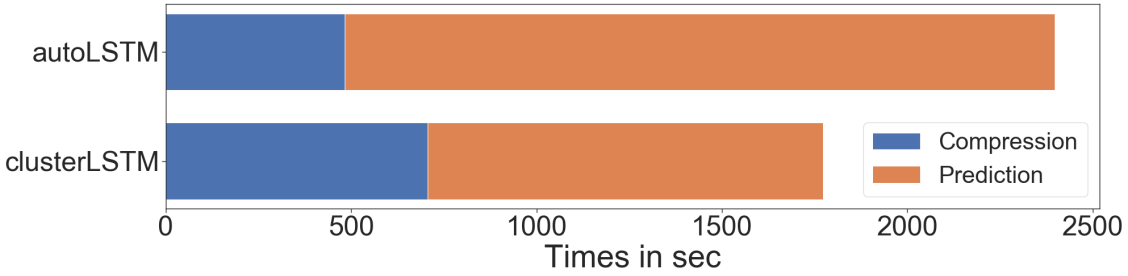


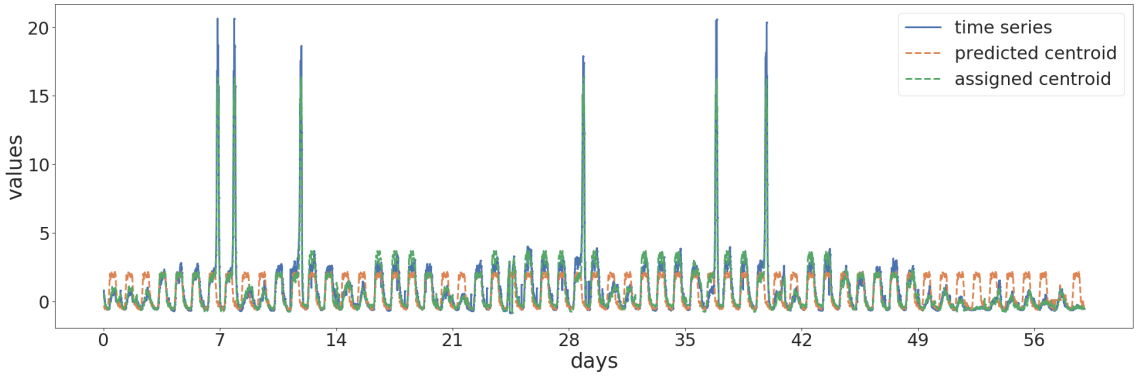
Figure 4.11: Comparison of run times between the clusterLSTM and the autoLSTM pipeline. The autoencoder is trained with 20 epochs. Both LSTM versions use a comparable number of epochs and a batch size of 16 (comp. Table 3.3). 72 cells are trained and predicted in each pipeline.

The memory usage depends on the size of the data set. The memory dimensions of our data are described in Section 4.2.2. Run times and memory usages of the two different predictions are presented in Table 4.10. Overall, the clusterLSTM requires approximately 3.8 GB, while the autoLSTM consumes around 3.6 GB.

Both compression methods are tested and measured on the same development environment at Ericsson. Nevertheless, it is important to consider that the environment is not a sandbox. Therefore, depending on the load of the server the CPU times can vary. Both LSTM solutions are tested on the same development server equipped with a GPU. As resources are shared between users, these measurements can vary depending on the number of active jobs on the server. In terms of memory usage the clustering requires noticeably more memory than the autoencoder. The LSTM approaches are very similar in regard to the memory usage. Analysing the CPU times presented allows the conclusion the complete pipeline is considerably faster than the benchmark as the efficiency of predicting an individual cell is much more efficient using the cluster labels. The clustering provides an advantage in data storage without loosing top much information as proven by the error between assigned centroid and true time series. To get a more detailed and accurate analysis of resource allocation and consumption, a sandbox system might be needed. Nevertheless, the presented performance results give an overall conclusive estimation and comparison.

In conclusion, the prediction results of both approaches demonstrate that the clusterLSTM prediction is competitive in terms of prediction error while requiring less computation. Further optimizations can be applied by improving the LSTM architectures and fine-tuning the cluster parameters. Moreover, it would be interesting to analyse the predictions for all 1 436 cells and explore whether the predictions are accurate for every type of cell. Moreover, as currently one individual model is trained for every cell, the cluster labels could be used to group and train similar cells based on their cluster labels. How this could be implemented is outlined in Chapter 3.7.1. Another improvement could be a separate prediction for every weekday, even though we assume that the LSTM is able to learn this sequence implicitly.

(a) Original time series



(b) Anomaly Detector

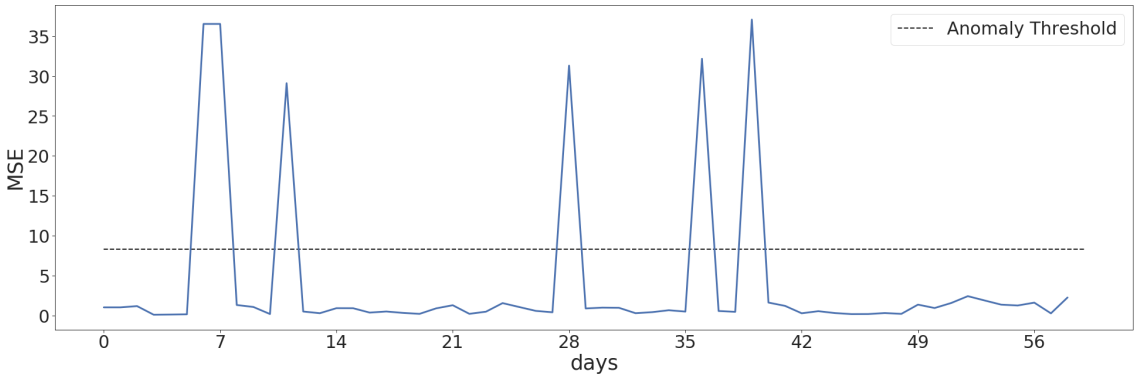


Figure 4.12: MSE between actual time series and predicted centroid. The dashed line is at averaged MSE *2. A day, whose MSE is above the dashed line could point to potential anomalies.

4.3.4 Anomaly Detection using Prediction Errors

Figure 4.12 a) presents the actual time series for 60 days of cell ABS and the predicted centroids and the assigned centroids. This cell is suitable to demonstrate the idea of using predicted centroids for anomaly detection as the cell shows clear deviations from regular time series pattern within the period of 60 days. At the same time, the predictions are very stable and accurate for the common days, because the MSE is between 0 and 1. Within the interval, seven high peaks can be observed that range from 0 to 25 whereas the common observed time series range from 0 to 5. Figure 4.12 b) below presents the MSE between the predicted centroid and the actual observed time series and the average MSE as a dashed line. This visualization demonstrates that the seven uncommon peaks can be identified indirectly by our method using the deviation between the predicted centroids and the actual time series.

By analysing and comparing a representative subset of the 72 cells predicted, we can conclude the following. Given that the calculated MSE over time is relatively stable and small in relation to the scale of the time series, we can rely on the prediction. If we can rely on the prediction, the approach of using large deviations from the average

observed MSE is a very promising approach to detect anomalies. The anomaly detection does not rely on a perfectly accurate prediction, but rather an indication for a clearly different shape or scale of the time series observed. Hence, a lossy compression can be used instead of a lossless compression. This anomaly detection can be used without any additional machine learning after the prediction and only requires the efficient clusterLSTM prediction. This approach could be tested on a labeled data set to validate the model. In general, it would be interesting to explore the results for the entire 1 436 cells in the data sets.

4.4 Conclusion

The evaluation of our compression baseline shows that Greedy Plain is an efficient algorithm to compress a moderate number of time series similar to the size of our data sets. Our results however indicate that this algorithm is performing worse for larger data sets, as the run time depends on the number of clusters. The specialised versions Greedy SF and Greedy SAX are more applicable when the data set becomes larger due to the candidate selection. This selection ensures a run time complexity that is mostly independent from the number of clusters. Especially, the candidate selection in Greedy SF can accelerate the clustering while still maintaining precision. Furthermore, it seems to be more robust than Greedy SAX as fewer single clusters are produced even though a similar *tau* was used. Greedy SAX shows a higher dependency on the number of generated clusters since the entries in the lookup table increase respectively. It was important to realise that despite an efficient computation, the algorithm was not creating the clustering results we expected in the beginning. This insight came from analysing various cluster results and exploring many characteristics of the compression, namely the number of clusters, cluster members, single clusters and many descriptive plots of many cells and their time series over different time intervals. This resulted in the awareness that different distributions of scales exist within our data set that could not be differentiated by our initial Greedy Plain algorithm. It was important to understand that a z-normalisation before the clustering does not ensure a good compression and according predictions. Therefore, we realized the need for a change of the *tau* logic. Implementing the magnitude adaptive clustering using a dynamic *tau* increased the compression rate considerably and contributed to more accurate clusterLSTM predictions.

We are confident that a compression using Greedy SF or Greedy SAX could be used to compress much larger data sets efficiently. By using a fixed value for the number of potential candidates the computing time depends mainly on the size of the data set, the hardware resources available and the desired compression rate. We cannot directly influence the hardware resources. The desired compression rate might depend on the use case for the data or the efficiency that wants to be achieved. As described in our results, even a high compression rate can lead to satisfying results that are noticeably close to the used benchmark. We assume that for larger data sets, potential candidates might be disregarded due to a considerably small number of candidates in respect to the size of the data set. Here, adjustments can be implemented to set the number of potential candidates in relation to the number

of clusters. This would most likely result in a lower performance by a small factor. Nevertheless, it could make the clustering more robust.

Regarding the clustering, larger data sets with more than one million time series could be used to analyse the behaviour of the candidate selections. At the same time, additional alternatives for compression could be compared with Greedy SF.

The clusterLSTM results suggest that this approach is very efficient and accurate within the scope of our data set. We assume that this model could be applied for any time series data set, where the data follows an approximately stable repeating pattern within a defined interval. We suppose that steady changes in amplitude can be learned by the model. In general, the quality of the predictions is highly dependent on the quality of the compression. Regarding our data set, a moderate variability in cluster labels for every cell is necessary to enable the LSTM to learn the different behaviour of weekdays. This type of prediction is highly efficient as a comparison to the autoLSTM shows. Even though the compression of the autoencoder and the clustering was very comparable, the clusterLSTM was approximately two times faster than the benchmark, while still maintaining a competitive precision.

The use case for anomaly detection demonstrates that prediction errors can be used effectively to identify anomalies for a subset of the predicted cells. This model allows some uncertainty in the prediction and could therefore be an extremely efficient way of identifying anomalies as no machine learning is needed additionally to the LSTM predictions.

5

Future Work

Grouping Cells for Prediction

Additionally to the idea presented in Section 3.7.1, cells can not only be grouped together through the similarities in cluster labels, but also by their geographical location. This can be achieved by retrieving this information about each cell in the network and group them by factors as population density.

Probabilistic Model for *tau*

For further studies about improving the current Greedy Clustering algorithm, one could explore the research provided by Blei and Frazier [3]. They developed the distance dependent Chinese restaurant process, a flexible class of distributions over partitions that allows for dependencies between elements. This could be translated to the problem of finding the different distance distributions within a time series data set and trying to derive boundaries for different clusters. Having a distance distribution, the expected distance distribution to other time series could be inferred for every time series. The distance distribution could be retrieved by computing all pairs of distances in a subset of the data set. It might be possible to provide a probabilistic model to study the Greedy Clustering algorithm.

Precomputing a set of different τ

A subset of the data can be used to find a set of different scales in values. A lookup table for τ can be added where depending on the value range of each individual time series a specific τ is chosen for the comparison. This could improve the computational complexity of the algorithm and save computation time.

Improve Greedy Clustering

A simple improvement to save computational time can be made in the second iteration of Greedy Plain, Greedy SF and Greedy SAX. The time series chosen as cluster representatives are not required be compared with all cluster centroids again since the distance to their own centroid is zero.

Moreover, the parameter *max candidates* can be implemented dynamically to allow more candidates as the number of clusters increases on the go during the clustering.

This could prevent overseeing potential candidates, especially for Greedy SF due to its performance advantages. For the Greedy SAX algorithm, a more efficient implementation in an architecture-close programming language might lead to a better performance. Additionally, research can be made to further improve the lookup table.

Another way of improving the performance of the Greedy clustering algorithm may be to investigate means of parallelising or distributing computation.

Bibliography

[1] Luai Al Shalabi and Zyad Shaaban. Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix. In *Proceedings of International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2006*, pages 207–214. IEEE Computer Society, 2006.

[2] Gabriel Alpsten and Sharan Sabi. *Prototype-based compression of time series from telecommunication data*. Master, Chalmers University of Technology, University of Gothenburg, 2019.

[3] David M Blei, Peter I Frazier, and Carl Edward Rasmussen. Distance Dependent Chinese Restaurant Processes. Technical report, 2011.

[4] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 58–67, 2010.

[5] Y. Collet. Zstandard - fast real-time compression algorithm, 2017.

[6] DeepLearning. LSTM Networks for Sentiment Analysis — DeepLearning 0.1 documentation, 2015.

[7] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast Subsequence Matching in Time-Series Databases. Technical Report 2, 1994.

[8] S. Gunderson. Snappy, 2011.

[9] JA Hartigan and MA Wong Journal of the Royal Statistical Society. Series C. Algorithm AS 136: A k-means clustering algorithm. *JSTOR*, 1979.

[10] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 6(2):107–116, nov 1998.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[12] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[13] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[14] Jessica Lin and Yuan Li. Finding Structural Similarity in Time Series Data Using Bag-of-Patterns Representation. Technical report, 2009.

[15] Simon Malinowski, Thomas Guyet, René Quiniou, and Romain Tavenard. 1d-SAX: A Novel Symbolic Representation for Time Series. In Allan Tucker, Frank Höppner, Arno Siebes, and Stephen Swift, editors, *Advances in Intelligent Data Analysis XII*, pages 273–284, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[16] Jason R. Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data, jun 2010.

[17] Kumar Molugaram and G. Shanker Rao. Analysis of Time Series. *Statistical Techniques for Transportation Engineering*, pages 463–489, jan 2017.

[18] Andrew Ng. CS294A Lecture Notes Sparse Autoencoder. Technical report, 2011.

[19] John Paparrizos and Luis Gravano. K-shape: Efficient and accurate clustering of time series. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015-May:1855–1870, 2015.

[20] Chotirat Ann Ralanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. Mining Time Series Data. In *Data Mining and Knowledge Discovery Handbook*, pages 1069–1103. Springer-Verlag, may 2006.

[21] David Salomon and Giovanni Motta. Handbook of Data Compression Fifth Edition Previous editions published under the title "Data Compression: The Complete Reference". 2010.

[22] Kukkong Sirisambhand and Chotirat Ann Ratanamahatana. A Dimensionality Reduction Technique for Time Series Classification Using Additive Representation. In Xin-She Yang, Simon Sherratt, Nilanjan Dey, and Amit Joshi, editors, *Third International Congress on Information and Communication Technology*, pages 717–724, Singapore, 2019. Springer Singapore.

[23] T. Warren Liao. Clustering of time series data - A survey. *Pattern Recognition*, 38(11):1857–1874, nov 2005.

A

Appendix

A.1 Prediction Results

cell	Greedy Plain		Greedy SF		Greedy SAX		best pred
	MSE true time series		MSE true time series		MSE true time series		
	pred	cluster	pred	cluster	pred	cluster	
AAC	0.473	0.164	0.448	0.157	0.637	0.252	stats
AAJ	1.375	0.100	1.443	0.101	1.923	0.334	plain
ABY	0.260	0.163	0.362	0.191	0.342	0.284	plain
ACC	0.594	0.126	0.663	0.148	0.433	0.132	stats
ACF	0.228	0.097	0.275	0.126	0.317	0.142	plain
ACN	0.705	0.157	0.636	0.198	0.755	0.261	stats
ACP	0.146	0.064	0.134	0.066	0.608	0.093	stats
ACU	1.026	0.148	0.937	0.193	1.201	0.210	stats
ADQ	0.433	0.254	0.402	0.251	0.612	0.287	stats
ADR	0.250	0.103	0.351	0.125	0.402	0.134	plain
ADV	0.518	0.100	0.460	0.131	0.495	0.198	stats
AEM	0.209	0.071	0.182	0.077	0.186	0.086	stats
AFE	3.312	0.488	4.398	0.496	2.977	0.823	sax
AGB	0.400	0.155	0.519	0.191	0.352	0.219	sax
AGC	0.443	0.131	0.342	0.141	0.420	0.231	stats
AGE	0.661	0.217	0.723	0.290	0.894	0.396	plain
AGU	0.177	0.088	0.257	0.128	0.276	0.147	plain
AHI	3.260	0.172	2.530	0.195	3.776	0.187	stats
AHQ	0.279	0.115	0.326	0.129	0.315	0.147	plain
AHS	0.988	0.191	0.932	0.260	1.023	0.346	stats
AII	0.213	0.067	0.209	0.088	0.266	0.115	stats
AIK	0.522	0.146	0.609	0.152	0.528	0.200	plain
AJY	1.127	0.230	1.017	0.344	1.221	0.402	stats
ALT	0.152	0.066	0.220	0.101	0.251	0.132	plain
ANT	0.772	0.126	0.531	0.138	0.811	0.240	stats
AOA	1.263	0.160	1.323	0.196	1.437	0.265	plain
AOY	0.407	0.140	0.332	0.153	0.415	0.172	stats
APW	0.375	0.184	0.459	0.190	0.527	0.264	plain

APX	1.913	0.069	1.043	0.079	1.057	0.113	stats
AQK	0.185	0.100	0.381	0.137	0.304	0.144	plain
AQL	0.733	0.178	0.590	0.170	0.427	0.251	sax
AQP	0.332	0.188	0.483	0.213	0.402	0.326	plain
ART	0.705	0.285	0.721	0.296	0.651	0.381	sax
ASB	4.823	0.290	5.700	0.285	5.117	2.494	plain
ATE	0.662	0.225	0.577	0.237	0.713	0.370	stats
ATT	0.589	0.130	0.784	0.127	0.812	0.216	plain
AUX	0.725	0.157	0.646	0.185	0.738	0.240	stats
AVC	0.831	0.104	1.014	0.129	0.608	0.153	sax
AVF	0.559	0.131	0.565	0.138	0.477	0.164	sax
AWG	1.656	0.256	2.407	0.409	2.213	0.329	plain
AWM	0.164	0.059	0.169	0.065	0.176	0.092	plain
AXF	0.780	0.143	0.665	0.172	0.596	0.233	sax
AXK	0.811	0.225	1.141	0.220	1.147	0.290	plain
AYH	0.446	0.315	0.417	0.286	0.659	0.280	stats
AYV	1.430	0.137	1.031	0.144	1.346	0.286	stats
AZF	1.925	0.107	1.818	0.110	1.530	0.237	sax
AZG	0.364	0.180	0.531	0.213	0.590	0.337	plain
BAH	1.423	0.121	1.666	0.133	1.162	0.125	sax
BAW	0.428	0.149	0.572	0.219	0.399	0.263	sax
BBE	0.621	0.241	0.604	0.287	0.885	0.385	stats
BBM	0.585	0.115	0.484	0.127	0.407	0.205	sax
BBQ	0.340	0.107	0.406	0.116	0.508	0.181	plain
BBV	5.029	0.458	3.259	0.450	6.700	0.948	stats
BCI	0.987	0.154	0.973	0.159	1.058	0.238	stats
BCU	1.546	0.179	1.138	0.186	0.890	0.187	sax
BCY	0.652	0.136	0.662	0.142	0.930	0.193	plain
BDB	0.382	0.126	0.529	0.144	0.432	0.183	plain
BFL	0.160	0.062	0.175	0.076	0.154	0.099	sax
BFT	0.173	0.061	0.165	0.074	0.171	0.094	stats
BGI	0.690	0.168	1.048	0.181	0.678	0.259	sax
BGL	0.416	0.105	0.298	0.111	0.493	0.158	stats
BHF	0.328	0.141	0.421	0.130	0.271	0.195	sax
BIA	0.299	0.073	0.189	0.087	0.169	0.099	sax
BJC	0.625	0.129	0.700	0.119	0.616	0.178	sax
BJL	1.232	0.070	2.294	0.070	1.333	0.077	plain
BKL	0.869	0.161	0.903	0.167	0.946	0.290	plain
BLG	0.980	0.125	0.927	0.144	0.897	0.119	sax
BND	6.235	0.496	6.302	0.551	9.428	3.198	plain
BNQ	1.925	0.631	1.996	0.673	2.169	0.977	plain
BNW	0.651	0.255	0.553	0.249	0.612	0.308	stats
BOD	0.605	0.156	0.641	0.170	0.648	0.227	plain
BOK	0.621	0.284	0.589	0.284	0.904	0.348	stats

Table A.1: Comparison of prediction results between Greedy Clustering, Greedy SF Clustering and Greedy SAX Clustering

MSE true time series

cell	autoLSTM		Greedy SF + clusterLSTM	
	prediction	reconstruction	prediction	cluster
AAC	0.328	0.064	0.448	0.157
AAJ	1.217	0.067	1.443	0.101
ABY	0.149	0.047	0.362	0.191
ACC	0.314	0.047	0.663	0.148
ACF	0.145	0.040	0.275	0.126
ACN	0.385	0.079	0.636	0.198
ACP	0.081	0.022	0.134	0.066
ACU	0.516	0.071	0.937	0.193
ADQ	0.221	0.080	0.402	0.251
ADR	0.177	0.043	0.351	0.125
ADV	0.199	0.036	0.460	0.131
AEM	0.100	0.031	0.182	0.077
AFE	2.715	0.171	4.398	0.496
AGB	0.161	0.056	0.519	0.191
AGC	0.236	0.050	0.342	0.141
AGE	0.487	0.096	0.723	0.290
AGU	0.163	0.037	0.257	0.128
AHI	0.361	0.107	2.530	0.195
AHQ	0.352	0.041	0.326	0.129
AHS	0.844	0.094	0.932	0.260
AII	0.215	0.035	0.209	0.088
AIK	0.431	0.053	0.609	0.152
AJY	0.639	0.071	1.017	0.344
ALT	0.205	0.027	0.220	0.101
ANT	0.255	0.058	0.531	0.138
AOA	0.805	0.060	1.323	0.196
AOY	0.230	0.057	0.332	0.153
APW	0.463	0.067	0.459	0.190
APX	0.134	0.031	1.043	0.079
AQK	0.180	0.044	0.381	0.137
AQL	0.236	0.058	0.590	0.170
AQP	0.175	0.051	0.483	0.213
ART	0.253	0.102	0.721	0.296
ASB	5.101	0.197	5.700	0.285
ATE	0.389	0.069	0.577	0.237
ATT	0.454	0.040	0.784	0.127
AUX	0.433	0.084	0.646	0.185
AVC	0.371	0.039	1.014	0.129
AVF	0.282	0.049	0.565	0.138
AWG	1.233	0.106	2.407	0.409
AWM	0.130	0.020	0.169	0.065
AXF	0.348	0.068	0.665	0.172
AXK	0.573	0.072	1.141	0.220

AYH	0.270	0.087	0.417	0.286
AYV	0.606	0.066	1.031	0.144
AZF	1.190	0.063	1.818	0.110
AZG	0.349	0.056	0.531	0.213
BAH	1.013	0.042	1.666	0.133
BAW	0.195	0.051	0.572	0.219
BBE	0.520	0.086	0.604	0.287
BBM	0.185	0.034	0.484	0.127
BBQ	0.220	0.037	0.406	0.116
BBV	0.790	0.151	3.259	0.450
BCI	0.606	0.050	0.973	0.159
BCU	0.708	0.071	1.138	0.186
BCY	0.735	0.061	0.662	0.142
BDB	0.252	0.059	0.529	0.144
BFL	0.131	0.021	0.175	0.076
BFT	0.167	0.026	0.165	0.074
BGI	0.668	0.069	1.048	0.181
BGL	0.222	0.045	0.298	0.111
BHF	0.191	0.058	0.421	0.130
BIA	0.187	0.032	0.189	0.087
BJC	0.166	0.046	0.700	0.119
BJL	0.530	0.035	2.294	0.070
BKL	0.535	0.048	0.903	0.167
BLG	0.764	0.047	0.927	0.144
BND	6.574	0.382	6.302	0.551
BNQ	1.117	0.217	1.996	0.673
BNW	0.702	0.106	0.553	0.249
BOD	0.386	0.051	0.641	0.170
BOK	0.400	0.070	0.589	0.284

Table A.2: Prediction results of a pipeline including an autoencoder and stacked LSTM in comparison to the clusterLSTM prediction using Greedy SF compression

MSE true time series

cell	Greedy Plain pred small LSTM	Greedy Plain pred large LSTM	Greedy SF pred small LSTM	Greedy SF pred large LSTM	Greedy SAX pred small LSTM	Greedy SAX pred large LSTM	best pred
AAC	0,461	0,473	0.468	0,448	0,520	0,637	large - stats
AAJ	2,363	1,375	1.500	1,443	1,457	1,923	large - plain
ABY	0,341	0,260	0,252	0,362	0,315	0,342	small - stats
ACC	0,509	0,594	0,536	0,663	0,602	0,433	large - sax
ACF	0,304	0,228	0,213	0,275	0,214	0,317	small - stats
ACN	0,745	0,705	0,703	0,636	0,765	0,755	large - stats
ACP	0,120	0,146	0,152	0,134	0,148	0,608	small - plain
ACU	0,665	1,026	0,961	0,937	0,814	1,201	small - plain
ADQ	0,550	0,433	0,375	0,402	0,375	0,612	small - stats/sax
ADR	0,385	0,250	0,244	0,351	0,334	0,402	small - stats
ADV	0,593	0,518	0,459	0,460	0,696	0,495	small - stats
AEM	0,169	0,209	0,222	0,182	0,207	0,186	small - plain
AFE	2,911	3,312	3,100	4,398	3,180	2,977	small - plain
AGB	0,357	0,400	0,443	0,519	0,565	0,352	large - sax
AGC	0,513	0,443	0,430	0,342	0,372	0,420	large - stats
AGE	0,946	0,661	0,797	0,723	1,071	0,894	large - plain
AGU	0,270	0,177	0,189	0,257	0,204	0,276	large - plain
AHI	3,531	3,260	2,041	2,530	2,363	3,776	small - stats
AHQ	0,325	0,279	0,341	0,326	0,347	0,315	large - plain
AHS	1,208	0,988	0,852	0,932	1,164	1,023	small - stats
AII	0,278	0,213	0,216	0,209	0,188	0,266	small - sax
AIK	0,567	0,522	0,599	0,609	0,777	0,528	large - plain
AJY	1,054	1,127	1,151	1,017	1,026	1,221	large - stats
ALT	0,238	0,152	0,140	0,220	0,260	0,251	small - stats
ANT	0,633	0,772	0,624	0,531	0,581	0,811	large - stats
AOA	1,459	1,263	1,249	1,323	7,012	1,437	small - stats
AOY	0,413	0,407	0,402	0,332	0,314	0,415	small - sax
APW	0,389	0,375	4,202	0,459	0,556	0,527	large - plain
APX	1,009	1,913	1,309	1,043	1,501	1,057	small - plain
AQK	0,276	0,185	0,216	0,381	0,304	0,304	large - plain
AQL	0,440	0,733	0,516	0,590	0,404	0,427	small - sax
AQP	0,400	0,332	0,303	0,483	0,383	0,402	small - stats
ART	0,600	0,705	0,580	0,721	0,685	0,651	small - stats
ASB	4,832	4,823	4,135	5,700	4,702	5,117	small - stats
ATE	0,719	0,662	0,444	0,577	0,544	0,713	small - stats
ATT	0,778	0,589	0,643	0,784	0,757	0,812	large - plain
AUX	0,668	0,725	0,710	0,646	0,535	0,738	small - sax
AVC	0,703	0,831	1,182	1,014	1,126	0,608	large - sax
AVF	0,479	0,559	0,549	0,565	3,887	0,477	large - sax
AWG	2,131	1,656	1,864	2,407	2,021	2,213	large - plain
AWM	0,157	0,164	0,177	0,169	1,858	0,176	small - plain

AXF	0,503	0,780	0,647	0,665	0,605	0,596	small - plain
AXK	1,122	0,811	0,834	1,141	1,187	1,147	large - plain
AYH	0,542	0,446	0,401	0,417	0,413	0,659	small - stats
AYV	0,984	1,430	1,112	1,031	1,229	1,346	small - plain
AZF	2,884	1,925	1,998	1,818	1,963	1,530	large - sax
AZG	0,515	0,364	1,293	0,531	0,475	0,590	large - plain
BAH	1,637	1,423	1,604	1,666	1,866	1,162	large - sax
BAW	0,528	0,428	0,341	0,572	0,641	0,399	small - stats
BBE	0,787	0,621	0,605	0,604	0,605	0,885	large - stats
BBM	0,514	0,585	0,605	0,484	0,476	0,407	large - sax
BBQ	0,469	0,340	0,327	0,406	0,410	0,508	small - stats
BBV	7,362	5,029	8,612	3,259	10,109	6,700	large - stats
BCI	0,688	0,987	0,895	0,973	0,953	1,058	small - plain
BCU	0,913	1,546	0,981	1,138	1,092	0,890	small - plain
BCY	0,825	0,652	0,651	0,662	0,603	0,930	small - sax
BDB	0,475	0,382	0,361	0,529	0,440	0,432	small - stats
BFL	0,186	0,160	0,166	0,175	0,176	0,154	large - sax
BFT	0,182	0,173	0,142	0,165	0,169	0,171	small - stats
BGI	1,684	0,690	0,828	1,048	0,754	0,678	large - sax
BGL	0,562	0,416	0,301	0,298	0,300	0,493	large - stats
BHF	0,272	0,328	0,317	0,421	0,412	0,271	large - sax
BIA	0,131	0,299	0,217	0,189	0,251	0,169	small - plain
BJC	0,635	0,625	0,611	0,700	0,461	0,616	small - sax
BJL	1,220	1,232	1,220	2,294	1,593	1,333	small plain/stats
BKL	0,713	0,869	0,858	0,903	0,816	0,946	small - plain
BLG	0,671	0,980	1,220	0,927	1,077	0,897	small - plain
BND	5,008	6,235	5,230	6,302	5,142	9,428	small - plain
BNQ	2,080	1,925	1,907	1,996	1,569	2,169	small - sax
BNW	0,531	0,651	0,580	0,553	0,536	0,612	small - plain
BOD	0,758	0,605	0,690	0,641	0,669	0,648	large - plain
BOK	0,834	0,621	0,547	0,589	6,669	0,904	small - stats

Table A.3: Comparison of predictions for the small LSTM architecture (2x 14 neurons in the hidden layer) and the larger architecture (2x 25 neurons in the hidden layer)

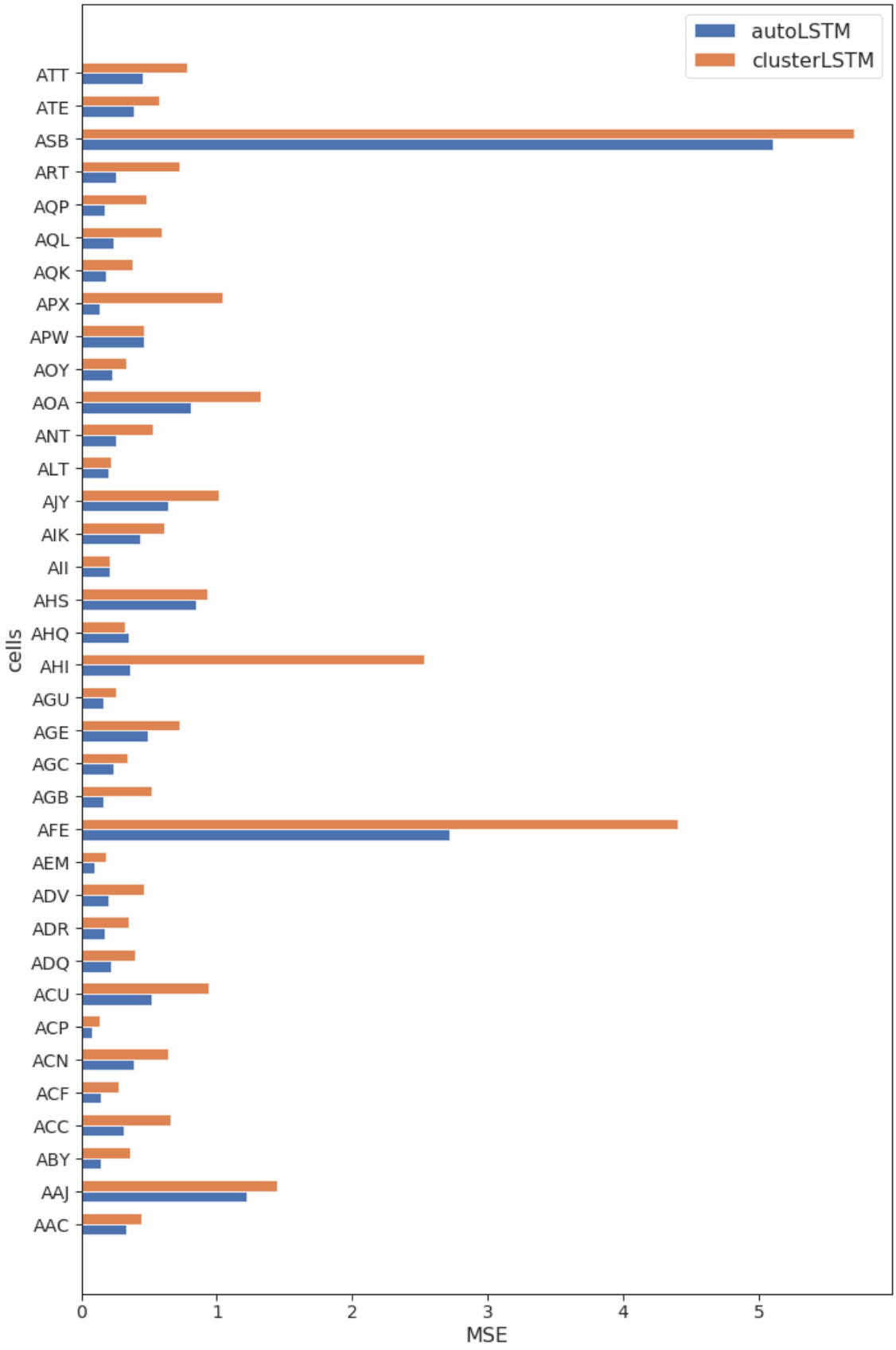


Figure A.1: Comparison of prediction MSEs for cell AAC - ATT

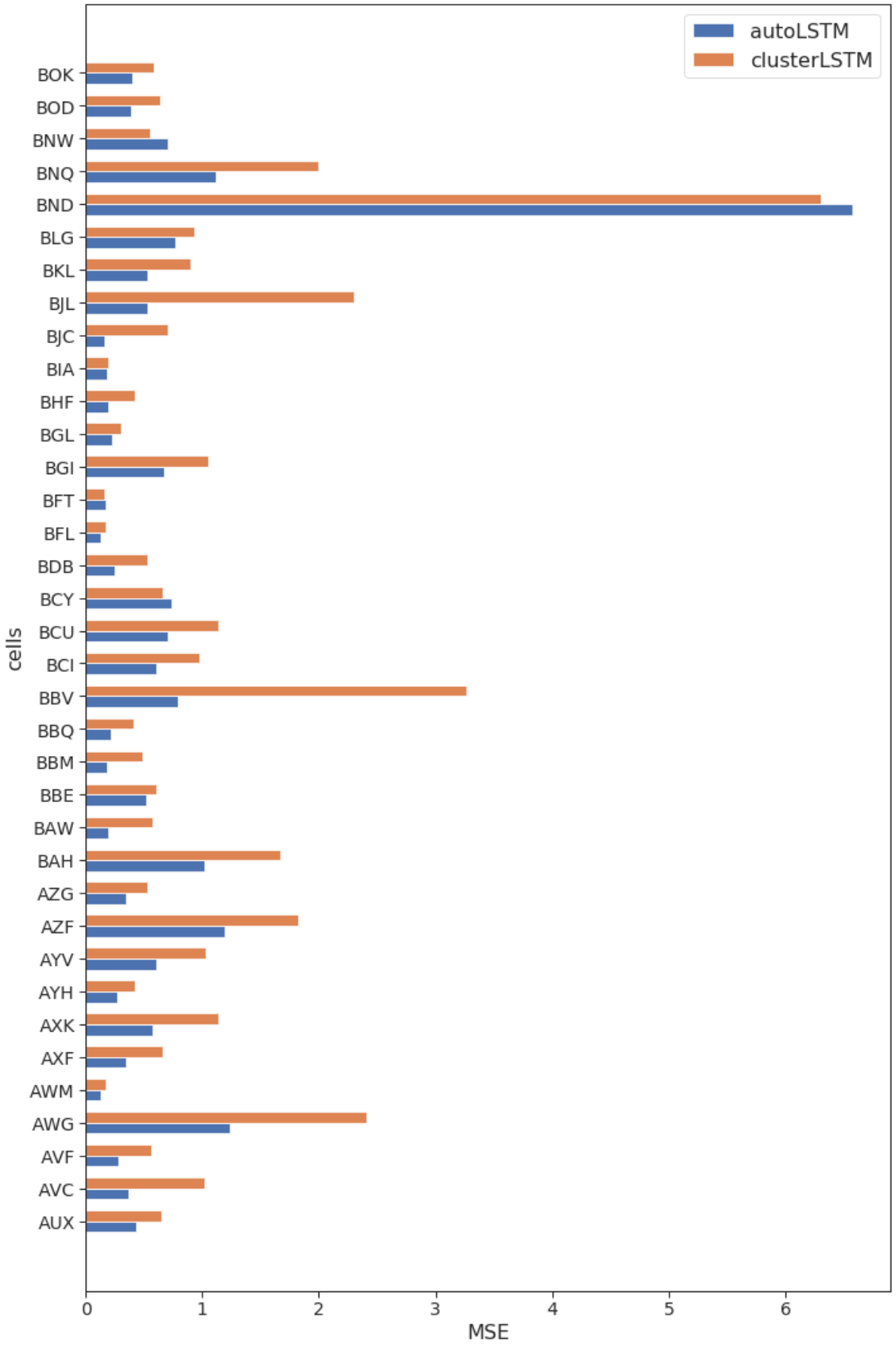


Figure A.2: Comparison of prediction MSEs for cell AUX - BOK

A.2 Evalution of Clustering

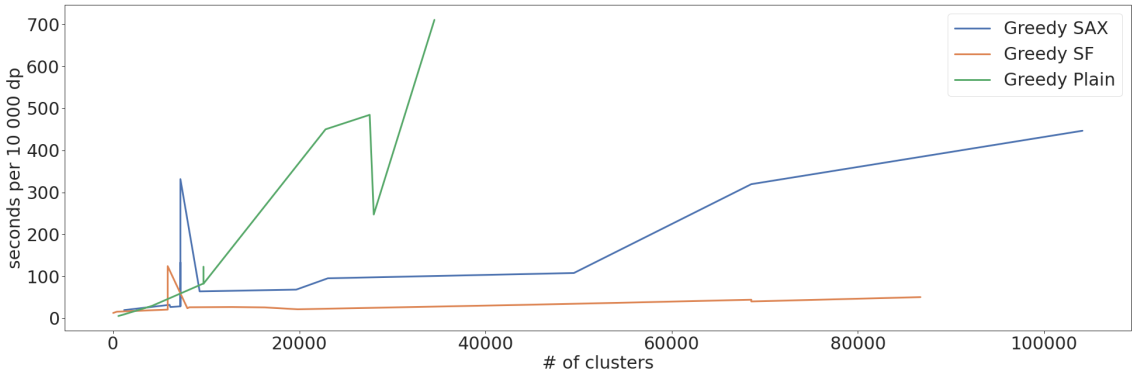


Figure A.3: Clustering times. Additionally, with higher number of generated clusters

```
Wrote profile results to benchmark_auto_lstm.py.lprof
Timer unit: 1e-06 s

Total time:     482.985 s
File: benchmark_auto_lstm.py
Function: train at line 28

  Line Hits          Time  Per Hit   % Time  Line Contents
==============================================================
    28                                           @profile
    29                                           def train(x, epochs=20):
    30   1  482985448.0 482985448.0  100.0     autoencoder.fit(x)
```

Listing A.1: Example output of line_profiler for function train