Delft University of Technology

Master Thesis

Predicting Near-Future Demand of Self-Storage Rooms

L. Overdevest (4374436)

Multimedia Computing Group
Department of Intelligent Systems

October 22, 2021

Predicting Near-Future Demand of Self-Storage Rooms

by

Lennart Overdevest

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday October 28, 2021 at 16:00 PM.

Student number:	4374436	
Project duration:	February 8, 2021 – October 28, 2021	
Thesis committee:	Dr. H. Wang,	Multimedia Computing Group (TU Delft)
	O. Fernández Robledo,	Multimedia Computing Group (TU Delft)
	Dr. S. Picek,	Cybersecurity (TU Delft)
	Ir. A. Schippper,	ALLSAFE

This thesis is confidential and cannot be made public until October 28, 2023.

Abstract

The value of data has increased enormously over the last couple of years. Many datasets contain valuable information that can, for example, be used to make forecasts. In this thesis, the dataset of a company in the self-storage industry is analyzed. The company offers customers rental storage facilities, such as lockers, rooms, containers at several locations in the Netherlands. The rooms, also called units, are characterized by three features: location, volume, and floor. The dataset contains for each storage facility the entire renting history of each self-storage unit. For the company, it is of interest to understand and predict the demand of its customers, such that it can flexibly adjust the prices of the rooms and services based on demand. Beyond, forecasting users' demand for various types of products is a common and essential problem in many different domains (e.g., recommendation systems, transportation systems).

This thesis aims to predict the demand for the next week by applying white-box and black-box models. The problem is represented as a temporal weighted bipartite network prediction problem. Specifically, the goal is to predict the network structure at a time T+1 based on the bipartite network observed at time $T-k+1$, $T-k+2$, ..., T, where k is an integer and needs to be optimized such that the prediction error is minimized. By analyzing the data in its temporal dimension, using the autocorrelation and cross-correlation among different storage locations, floors and volumes, it was shown that the autocorrelation is high and the cross-correlation is low. This suggests that the temporal bipartite network is possibly predictable.

We have explored different state-of-the-art predictive techniques. Markov chain model, LSTM, and ConvLSTM have been selected because of their fundamental difference in the way they learn and predict. A performance comparison is given where the techniques have been applied on the storage data, and it shows that LSTM outperforms the Markov chain and ConvLSTM based on the following evaluation metrics: RMSE, MAE, and accuracy. According to our dataset, higher predictability was achieved when only the data of a single link was exploited. The Markov chain and the LSTM utilize the information of a single link to predict. On the contrary, the ConvLSTM utilizes the information of the entire network to predict. The low cross-correlation between the links explains why the LSTM outperforms the ConvLSTM. The ConvLSTM tries to capture spatio-temporal dependencies, while this, in general, does not contain much valuable predictive information. Thus, the model is introduced to more noise, making it harder to predict accurately. The LSTM also outperforms the Markov chain model, which is used as a baseline method. This proves that it is beneficial to use a complex deep learning model for this dataset to predict. However, the Markov chain performs comparable to the ConvLSTM, showing that a black-box model does not always outperform a white-box model. This emphasizes that the most suitable predictive algorithm depends on the statistical properties of the dataset.

The theoretical upper bound of the predictability of the network is computed. It is the upper bound that can be used to compare the realized performance to the maximum achievable prediction performance for any predictive algorithm. The difference between the performance of the best performing algorithm to our dataset, LSTM, and the theoretical upper bound is still large, indicating that there is still room for improvement.

Acknowledgements

First of all, I would like to sincerely thank the Multimedia Computing Group and ALLSAFE for giving me the opportunity to pursue a study in a field of my interest. It allowed me to get hands-on experience and enabled me to put theory into practice. The MSc courses at Delft University of Technology have been the backbone of the results gained in this thesis.

My supervisors Dr. Huijuan Wang and Omar Fernández Robledo have thoroughly helped me during the previous nine months. Despite the severe working from home conditions due to COVID19, they were always available and put a significant of time into me, which I am very thankful for. Besides that, I want to thank my colleagues at ALLSAFE and in particular Ir. Arjan Schipper for providing in-depth feedback from a company perspective, as well as, giving advice on how to increase the quality of my project.

Lastly, I would like to thank my family, girlfriend, and friends for their support during my studies. Their faith in me helped me achieve my goals.

Lennart Overdevest
Delft, 2021

Contents

List of Figures

List of Tables

1 Introduction

Many real-life complex systems can be modeled as a network. A network consists of nodes and edges, also called links. These networks range from social networks (e.g., Facebook, LinkedIn), where a user represents a node, and an edge denotes a type of friendship between two users, to transportation networks where train stations represent the nodes and an edge indicates the connection between the train stations. If the edge has a weight, the network is called a weighted network. Real-life networks change over time, meaning a relationship between nodes can vary over time. Therefore, an extra dimension is added to the structure of the network. Instead of having a static network consisting of nodes and edges, a dynamic network contains a set of nodes and edges that are time-dependent [3].

Analyzing the way a network evolves can be of high value [4], [5]. There are many interesting properties of a network that can be examined, and an example is detecting clusters [6]. In this project, the focus is on predicting the network's structure in the future. This problem is also called the temporal link prediction problem or the temporal network prediction problem. The goal is to predict the network structure at a time $T+1$ based on the network observed at time $T-k+1$, $T-k+2$, ..., T, where k is an integer and needs to be optimized such that the prediction error is minimized.

Because more and more real-life complex systems can be modeled as a network, the temporal link prediction problem has received much attention. It is researched in many different domains, such as bioinformatics [7], epidemiology [8], recommendation systems [9], and social networks [10]. In these domains, it is often the goal to predict the occurrence of new links in the network. However, the goal in this project is to predict the network structure (weights) for the existing edges, and therefore, slightly differs. Although, it emphasizes the importance of the temporal link prediction problem and why it is one of the main research areas in network science.

Another network of interest is the bipartite network. This network is a particular type of network where all nodes can be divided into two separate groups such that the nodes within the same group cannot be directly connected. A well-known example of such a graph is the user-item network in the E-commerce field, where users can buy several items but users can not be directly connected. In these types of networks, it is often the goal to predict whether a node will connect to another node in the future. Again taking the E-commerce network as an example, it would be interesting to know which and when a user will buy an item.

The dataset used in this project is from a company in the self-storage industry, which offers customers rental storage facilities, such as lockers, rooms, containers, usually on a month-to-month basis. This thesis will explore which network fits best on the dataset that the storage company provides.

Predicting future information about renting storage rooms for the company can be seen as a problem that can be modeled as a temporal network prediction problem. The network consists of two different types of nodes. The first set of nodes represents the location of the storage facilities of the company. The other set of nodes represents different types of storage rooms, each of which has a unique combination of volume and floor features. Since the locations have no specific relation, there will be no links/edges between the locations. The same is true for the relation between the types of storage rooms. Therefore, the problem in this project can be modeled as a temporal weighted bipartite network where the nodes on the left side represent locations and the nodes on the right side types of storage rooms. An edge exists between a left-side node and a right-side node if the location has at least one storage room of the defined volume and floor. The weight of the edge represents the demand for a specific type of storage rooms (with the same volume and floor) at a particular location. In Figure 1.1, a small example of such a network is shown with four locations (green nodes) and four types of storage rooms (white nodes) defined by their volume-floor combination. As already mentioned, it is interesting to analyze how this network evolves. Therefore, the problem of predicting future weighted temporal links will be researched on a bipartite weighted network.

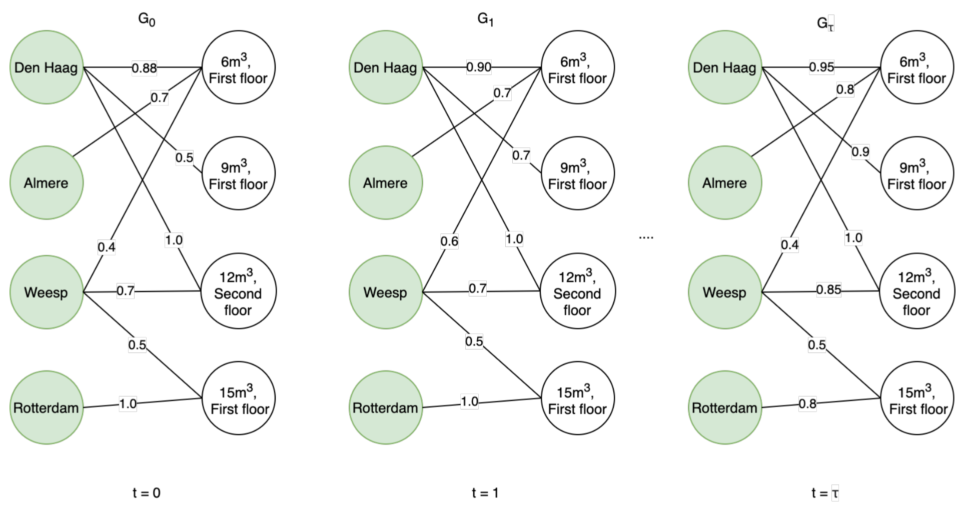


Figure 1.1: Example of a temporal weighted bipartite network.

1.1. Motivation

As already highlighted above, the value of data has increased enormously. ALLSAFE, a company that rents storage rooms of different sizes all over the Netherlands, owns such a valuable dataset. This dataset contains for each storage room, also called a unit, the exact moment a unit is rented. In Figure 1.2, per location (in total 38), the number of years of data is shown. As can be seen in the figure, several years of data are available for most locations. This (temporal) dataset can be analyzed to see whether it is possible to predict future demand based on historical data. Therefore, the main focus of this thesis is to exemplify how to analyze the temporal link prediction problem on a real-world dataset. The predictions can, for example, be used to flexibly adapt the prices of the units and services based on demand.

In the literature, many approaches have been proposed to solve the temporal link prediction problem, which will be analyzed in Chapter 3. However, many of these approaches are tested on classical datasets. These typical datasets include, for example, the author and conference network and the E-commerce network containing the relationship between user and items. The goal of these approaches is often to predict the occurrence of new links. However, in this project, the goal is to predict the weight of the edge (demand). Besides that, it will be demonstrated how to apply the different predicting algorithms on a real-world dataset. Prior to that, data preprocessing will make sure that the input data is selected consistently over time and across the locations, such that it is in the desired format for the input of our prediction models. The performance of these different methods will be compared to each other, which should prove what algorithm is best applicable to this real-world dataset.

Analyzing a real-world dataset always introduces difficulties, such as missing data or unstructured data. Thus, in addition to forecasting demand, another critical challenge is the data preparation to create a complete and structured dataset that can be used for the predictions. This structured dataset is then used to research whether modeling the problem as a network provides many benefits compared to analyzing the links of the network individually.

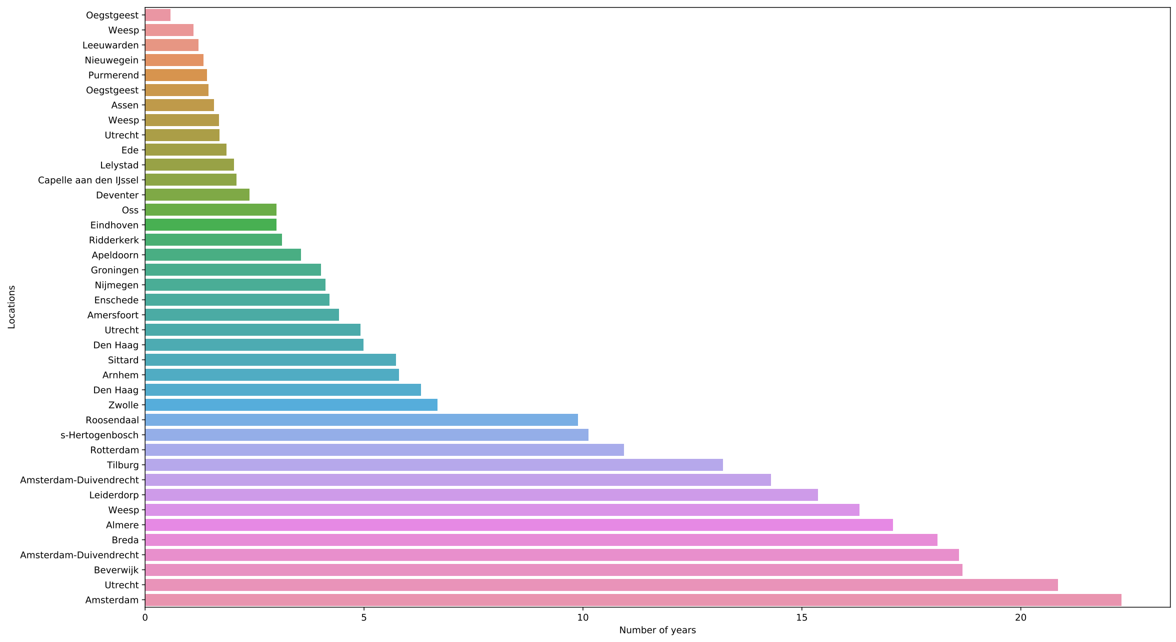


Figure 1.2: Histogram that shows for each individual location (site) the total number of years of data being available.

1.2. Problem Statement

The problem can be formulated as the temporal link prediction problem. The goal of this problem is to predict the network structure at a time $T+1$ based on the network observed at time $T-k+1$, $T-k+2$, ..., T, where k is an integer and needs to be optimized such that the prediction error is minimized. In this project, the network is a temporal weighted bipartite network, where each node on the left side represents a location, and each node on the right side represents a set of units that can be distinguished by equal volume and floor. The weight of an edge represents the demand/occupancy rate for a location and a specific type of units defined by its volume and floor at a certain point in time. The occupancy rate is the ratio of rented units over the total units at the given location and of the given type specified by volume and floor.

1.3. Research Questions

The main research question has been formulated as follows:

To what extend is it possible, based on historical data $T-k+1$, $T-k+2$, ..., T, to predict the demand at time $T+1$, for a given type of storage, at a given location?

The main research question can be narrowed down into multiple sub-questions. Each of these questions is addressed in a chapter or section later in the thesis.

- *Does the data consist of any patterns, if so, which patterns indicate the predictability of the network?*
- *If yes, which type of technique is most suited for the demand prediction problem, and why?*
  - *Do black-box methods outperform white-box methods?*
- *What is a suitable metric to evaluate the predictions made by the models?*
- *What is the impact of the number of time steps k used to predict the performance of the model?*

1.4. Proposed Solution

We preprocess the data to represent the problem as a temporal weighted bipartite network. Data analysis should then lead to finding patterns that will give insights into the predictability of this dataset. Based on this analysis, it will be proven that it is possible to forecast the demand for the different locations and units. Besides that, it also supports the motivation of selecting and applying the following three predictive algorithms to predict the occupancy rates of the units:

- Markov chain model
- LSTM
- ConvLSTM

The Markov chain is a statistical method that predicts the future state based on the previous state. In this project, the states of the Markov chain represent the demand ratio of the units. The transition matrix of the Markov chain contains the probability of transitioning from one state to the next state. These probabilities are computed based on the data. The future state is then predicted based on the input state and the transition matrix. This algorithm predicts solely based on the transition probabilities and can therefore be classified as a white-box model. Because of this, the predictions of the model can be explained, which is not possible for the black-box models.

The LSTM is a deep learning model and can therefore be classified as a black-box model. The Markov chain and LSTM approach the problem from a link perspective, which means that the input of these models is the data of a single link in the network. Based on the data of an individual link, they learn and predict the future state of this link. This process is repeated for each link in the network.

Lastly, the ConvLSTM is an extended version of the LSTM. The ConvLSTM takes several graph snapshots as input and thus utilizes the information of multiple links in the network to learn and predict the demand ratio of each link. This method could therefore use the correlation between the different links in the network to predict. It is worth mentioning that it predicts the demand for the entire network instead of a single link.

By comparing the differences in performance, we can explore whether a model that utilizes the information of the entire network outperforms a model that utilizes only the information of a single link to predict. The evaluation metrics to evaluate the performance of the models are Mean Absolute Error, Root Mean Squared Error, and accuracy.

1.5. Contributions

Below, a summary is given of the main contributions of this thesis:

- Creating a preprocessing pipeline to generate a structured and consistent dataset from a real-life dataset. From this dataset, a temporal weighted bipartite network is constructed.
- Because the temporal data of each individual link in the network can be represented as a time series, basic time series analysis techniques (probability density function, autocorrelation function, cross-correlation function) are used to analyze the temporal data. The results of the data analysis can be used to analyze the predictability of the data and to explain the prediction performance of each model.
  - The probability density function is used to plot the demand distribution to get an insight into the impact of the volume, floor, and location on the demand.
  - The autocorrelation is computed between each time series and a lagged version of the time series itself. In this way, it can be computed how correlated future demand is with previous demand. The higher the autocorrelation, the higher the predictability of the network.
  - The cross-correlation can indicate how non-trivial the temporal change is of the demand of the different locations and types of units. Besides that, the cross-correlation can also be used to explain the prediction performance when the performance of the models that utilize the information of the entire network is compared to the performance of the models that utilize only the information of a single link to predict.
- Selection and application of three state-of-the-art predictive algorithms and an elaborate explanation on why these models are selected. Also, a detailed study on the hyperparameters is provided, uniquely defined for each model.

- Performance analysis of the three implemented methods on a real-life dataset. This analysis is also used to analyze whether approaching the problem from a network perspective provides any benefits compared to using only the information of a single link to predict.
- Applying a recently published theoretical framework on a real-life dataset that can quantify the predictability of a temporal network. By comparing the theoretical upper bound of the predictability of the network with the performance of the models, it is explored how these predictive algorithms perform compared to the theoretical upper bound.

1.6. Thesis Structure

The thesis is structured as follows. First, the theoretical background needed for this report will be elaborated on in Chapter 2. This includes a brief introduction about the dataset used in this project, a mathematical formulation of the problem, and a list of definitions used in this report. Then, in Chapter 3, the approaches proposed in the literature will be categorized and compared. In Chapter 4, an in-depth explanation is given of how the data is preprocessed to generate a complete and structured dataset. Furthermore, it shows how the problem can be modeled as a network and contains the data analysis results. Also, an analysis is done with respect to the units and the demand, which showcases the autocorrelation and cross-correlation outputs for different combinations.

Chapter 5 gives an overview of the applied methods. It discusses the model's input and a high-level overview of the working of the models. Chapter 6 evaluates the performances of the models. This includes comparing the RMSE and MAE of the different models and comparing the upper bound of the predictability of the network with the accuracies of the different models. Finally, Chapter 7 concludes the results of this project. Also, the limitations and future improvements will be discussed in this section.

2 Background

In this chapter, the reader is introduced to the background information which is used in this thesis. First, a brief introduction is given about the company. Then the input data is described, followed by a section describing how a network is mathematically formulated. Then a list of definitions is given with their exact meaning.

2.1. ALLSAFE

ALLSAFE is a company that offers storage space for people who need it. Customers can rent a unit ranging from XS (1-3m^3) to XL (55-120m^3). The units can be rented for personal reasons, for example, when two people want to live together but their new house is not ready yet. They are temporally living in a smaller house, and in order to store their items, a unit is rented until their new house is built. It can also be used for business reasons. Two examples of this are storing items of a webshop and saving archives of an office.

ALLSAFE has been around for over 20 years. The first location was opened in Amsterdam in 1999. By the time of writing, the number of locations had grown to 38 locations located all over the Netherlands. As can be seen in Figure 2.1, the locations (marked with a blue marker) are mainly centered around big cities.

2.2. Data

In this section, a brief overview is given of the data used in this project. The company's data is saved in a SQL [11] database and could therefore be exported to several commas separated values (CSV) files. Multiple tables were exported, but the most important one is the table containing the complete history for each unit. By the time of writing, the table contains 877,164 records. As can be seen in Table 2.1, for every unit it contains the moment when a unit switches from a status (FromUnitStatusID) to another status (toUnitStatusID). From the second row in the table, it can be seen that on 2008-06-23 unit with ID 5211 was rented because the StatusID switches from status 0 (free) to 1 (in use). In general, the sequence of the status of a unit is as follows:
waiting for delivery (-1) → free (0) → reserved (5) → in use (1) → unsubscribed (7) → free (0).
By the use of the ContractID a unit is linked to a customer. Analyzing the characteristics of the customers can be considered out of the scope of this research. Therefore the ContractID will not be used in this project.

Table 2.1: Snapshot of the CSV file containing the entire history of all units.

UnitLogID	UnitID	FromUnitStatusID	ToUnitStatusID	Description	ContractID	TimeStamp
196	5213	0	5	New reservation		2008-06-11 16:00:44
204	5211	0	1	New contract	31757	2008-06-23 17:50:36
205	5211	1	2	Customer needs to pay	31757	2008-06-24 11:36:06
206	5314	0	1	New contract	31758	2008-06-30 18:10:42
207	5252	0	5	New reservation		2008-07-02 16:16:24

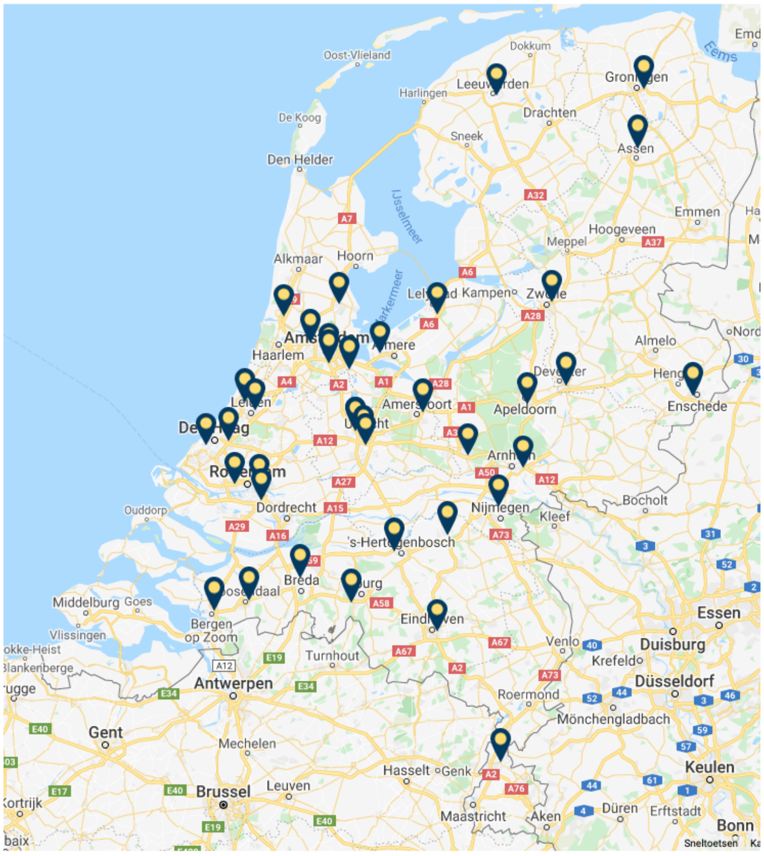


Figure 2.1: Map showing the demographic location of all ALLSAFE locations.

It is important to note that the unit log (containing the entire renting history of each unit) contains some inconsistencies. An example of this is the sequence of the status of a unit discussed above. The statuses of the units which were built at the very early stage do not match this pattern. For these units, a record was inserted when the unit was rented for the first time. Because the units were not inserted into the unit log when rentable, the exact moment the units were rentable is undefined. This means that demand for these units is not 100 percent traceable because it is unknown how long it took before the units were rented after being marked as rentable. In Section 4.1, this problem and the solution to this problem will be discussed in more detail.

2.3. Graphs

From the data described in the previous section, a temporal weighted bipartite graph could be created. A temporal weighted bipartite graph is a sequence of graph snapshots $G = \{G_1, G_2, .., G_\tau\}$, where $G_t = (U,\ V,\ E,\ W_t)$ with all time steps defined by $t \in \{0,\ 1,\ 2,\ ..,\ \tau\}$ and τ being the current time step and W_t is the set containing the weights of the edges. U and V represent the different types of nodes (locations and unit types, respectively) and E the set of edges. As can be seen, the set of nodes and edges do not depend on time. This is because a subset of the data is chosen such that the number of units does not increase over time. In Section 4.1.2 this will be explained in more detail. On the other hand, the weights W_t do depend on time, where the subscript indicates for which time step t the weights are defined. Assume U contains m nodes, and V contains n nodes, then the graph G_t can be represented by an adjacency matrix $A_t = \left[A_{m\times n}\right]$.

Thus, if there exists an edge between node i and j ((i, j) $\in E$) with weight W_t(i, j) then A_t(i, j) = W_t(i, j). The network is undirected, thus A_t(i, j) = A_t(j, i). In Figure 1.1, a small example of such a network is shown.

Now that the problem is defined more mathematically, the goal of the temporal link prediction can be defined as follows: Given the previous k adjacency matrices (A$_{\tau-k+1}$, A$_{\tau-k+2}$, .. A$_{\tau}$) the goal is to predict the adjacency matrix A$_{\tau+1}$, where k is the number of snapshots used to make the prediction. Therefore, the parameter k is of utmost importance for making predictions and needs to be set correctly according to the temporal history in the network, which is still to be determined.

2.4. Definitions

Throughout this report, several definitions will be used. In order to have a clear understanding of these definitions, the most important terms are defined below:

- *UnitLTG* - a set of units that have the same location, type, and group. In other words, the units with the same location and volume and are located on the same floor. For example, UnitLTG (Leiderdorp, 6m^3, First floor) means all units at the Leiderdorp site with volume 6m^3 which are located on the first floor. For this example, the set contains 21 units.

- *UnitLT* - a set of units which have the same location and type (volume). For example, UnitLT (Leiderdorp, 6m3) means all units at Leiderdorp with volume 6m^3. In total there 48 individual units with volume 6m^3 at Leiderdorp.

- *Demand* - the demand, also called occupancy rate, defined as D at time step t can be computed with the following formula,

$$D(t) = N_{occupied}(t)/N_{total} \tag{2.1}$$

  where $N_{occupied}(t)$ represents the total number of occupied units at time step t and N_{total} the total the number of units for a specific UnitLTG or UnitLT and $0 \leq D \leq 1$. It is worth noting that it is assumed that N_{total} does not change over time, more information about this can be found in Section 4.1.2. To show how the demand is calculated an example is given. Assume, we want to calculate the demand for a specific UnitLTG at time step 10, the number of occupied units for UnitLTG (Leiderdorp, 6m^3, First floor) was 18. Therefore, the demand is equal to D(10) = $\frac{18}{21}$ = 0.86.

3 Related Work

In this chapter, a high-level overview will be given of the relevant approaches proposed in the literature. This chapter aims to familiarize the reader with the different techniques used to solve the temporal link prediction problem. Firstly, the traditional methods will be discussed. Traditional methods can be defined as approaches that are well-studied and have existed for a long time. Then, an overview of the learning methods is given, followed by an overview of the approaches that focus on bipartite networks.

3.1. Traditional Link Prediction Methods

The approaches evaluated in this section are frequently used in the literature and applied in many different domains. The first approach is the Markov chain model [12]. This approach is used as a baseline method in [2] which focuses on defining a metric for the predictability of real temporal networks which can be used in any domain. Markov simplifies the problem by assuming that the next state only depends on the present state. In [2], a Markov chain model is created for each link in the network. The transition matrix, created using the training data, is used to predict given the input state. In this thesis project, also a transition matrix could be created for each link. However, the main difference would be that the states of the transition matrix represent the demand ratio instead of the destination node of a link. Thus, instead of predicting the destination node of a link, the goal is to predict the next weight of an edge which is equal to predicting the demand. This is an interesting approach to use as a baseline method because it simplifies the problem. Therefore, we can analyze what the influence is of the simplification on the performance of the model. Before applying this method, an analysis needs to be done to see whether the demand of the next time step depends on the demand of the previous time step. The autocorrelation function is a common technique that can be used to detect these patterns.

The second category contains the approaches that are based on matrix factorization and tensor factorization techniques. Matrix factorization is a technique used to capture the underlying patterns in a matrix by decomposing the matrix into the product of two lower dimensional matrices and is proven to be successful in many different domains (e.g., neuroscience [13], E-commerce [14], social network analysis [15]). Dunlavy et al. [16] used the above-defined techniques to predict links in temporal networks. In their paper, the performance of matrix factorization is compared with tensor factorization. The main difference between these two techniques is the dimensionality of the input matrix. Since matrix factorization can only deal with a matrix of the form $m \times n$, the network structures (matrix slice) over time need to be collapsed into a single matrix. On the contrary, tensor factorization can handle a matrix of three dimensions, $m \times n \times t$, where $m \times n$ represents the network's structure at a specific moment in time and t the number of time steps. It was proven that the tensor factorization approaches slightly outperform the matrix factorization approaches.

In [17], matrix factorization is used to combine information from multiple sources to predict the probability that a new link will occur. Since the paper mainly focuses on predicting the probability of the occurrences of new links, it does not precisely match this project's goal because no new links can occur in the network. However, the paper highlights the possibility of combining information from different sources (content information, structure of the network) using matrix factorization. This could be of importance during this project when additional information is used when making the prediction. For example, the housing demand in a specific region could be a good indicator for the demand for the storage facility in that region. Then using a similar approach, as proposed by Gao et al. [17], can be implemented such that the information of multiple sources can be combined.

Wenchao et al. [18] used matrix factorization to model the evolution of the edges in a network by taking into consideration the correlations and interactions with adjacent edges. In contrast to the approach proposed in [17], this approach can predict the weight of the edges and therefore matches with the goal of this problem. It is also proven that Wenchao's approach outperforms the technique proposed in [16] on several datasets.

In [19], an approach called Graph Regularized Nonnegative Matrix Factorization algorithm (GrNMF) is proposed. As the name already emphasizes, it uses an extended version of the matrix factorization technique. Ma et al. regularize the networks from time 1 to t-1 to factorize the network at time t. Using nonnegative matrix factorization, two (nonnegative) matrices are found by optimizing an objective function. The product of these two matrices approximates the original network.

Based on the approaches discussed above, it can be concluded that matrix and tensor factorization are often applied to make predictions in networks. Wenchao's approach [18] also takes the correlation between adjacent edges into account, which might contain valuable prediction information, and thus, can help improve the performance of the model. In order to find out if this correlation is indeed valuable, the cross-correlation function can be used to test the correlation between the edges (demand) in the network. In Section 4.3, it will be tested if any correlation between the demand of the different types of units and locations exists. If this is the case, this approach is a suitable method. Combining the information of different sources using the method proposed by Gao et al. is an interesting and promising approach. However, because of limited time, this is out of the scope of this project. Also, a different dataset needs to be available, which is not the case. As mentioned before, the Markov Chain model can be used as a baseline method. However, other approaches would probably outperform this method because of the oversimplified assumption made by this method.

3.2. Self-Learning Link Prediction Methods

Artificial intelligence, machine learning, and deep learning have all received much attention last decades. In network science, several of these techniques are used to analyze complex networks. An interesting forecasting approach with relatively more memory from previous time instances is the Long Short-Term Memory network, in short LSTM [20]. LSTMs have proven to be very successful and are applied in almost any domain that contains any form of sequential data (e.g., financials [21], natural language processing [22]). The goal of an LSTM is to memorize long-term information. The flow of information is regularized by an internal mechanism that defines the importance of the information. A variation of the LSTM is the Convolutional Long Short-Term Memory network (ConvLSTM) [23].

The main difference between ConvLSTM and LSTM is the number of input dimensions. As LSTM input data is constrained to one-dimensional input data, and is, therefore, not able to capture spatio-temporal dependencies while predicting because a network can not be modeled as a one-dimensional input vector. Because of this, the LSTM cannot use the correlation among links in the network, which could be of high value when making predictions. The ConvLSTM, however, can handle multiple dimension input and is used in predicting future networks. This technique uses the convolution operation. To summarize the fundamentals of the convolutional operation, a filter, also called the kernel with a predefined width and height, is shifted over the input matrix to create a new matrix called the feature matrix. This feature matrix is updated during the training phase and contains information of neighboring cells of the input matrix, which is used to predict. In this way, the ConvLSTM can capture spatio-temporal dependencies contained in the input matrix. In [24], this technique is used to predict the travel demand to optimize the traffic resources. In [2], a ConvLSTM is used to predict the links in 18 real temporal networks (e.g., mailing network, transportation network). It was shown that the ConvLSTM only slightly outperforms the Markov chain model. An advantage of ConvLSTM compared to, for example, the Markov chain model discussed earlier is that the ConvLSTM takes the correlation between links into account.
Another promising technique used to predict the structure of temporal networks is based on Graph Convolutional Networks (GCN) [25]. GCN is a variant of the deep learning technique Convolutional Neural Network (CNN) [26]. CNN has proven to be very successful in analyzing Euclidean input data, such as images [27] [28]. However, traditional CNNs cannot handle non-Euclidean input data, and therefore GCNs have received much attention thanks to their ability to handle non-Euclidean data. Both Zhao et al. [29] and Ge et al. [30] used GCNs to predict the traffic demand of a traffic network which has some similarities with predicting the occupancy rate of the units in this study. Besides the temporal dependence (e.g., peak hours in the network

on working days) in the traffic network, it also contains spatial dependence. Therefore these networks are called spatio-temporal networks. Spatial dependence can be seen as a strong influence on the traffic volume of adjacent roads. In this project, it needs to be analyzed if the network also contains any spatial dependence. It could be reasoned that the occupancy rate of two different units with an equal location influences each other. For example, if one type of unit is fully occupied at a specific location, the demand for other units at this location could increase. However, this assumption needs to be verified and will be analyzed in Section 4.3.

Zhao et al. proposed a Temporal-Graph Convolution Network (T-GCN) model, which can be divided into two parts. First, an GCN is used to capture the topological information. Then, the Gated Recurrent Unit (GRU) is used to capture the dynamic change of the network over time. GRU is a similar method as LSTM. However, GRU is faster because fewer parameters need to be optimized. Ge et al. proposed a model called Graph Temporal Convolutional Network (GTCN). Besides the traffic data, this model can handle external data, such as social factors (e.g., holidays, peak hours), road network structure (e.g., bridges), and points of interest (schools, restaurants) which can influence the traffic volume of the roads nearby. This external data is not available in this project. However, this approach emphasizes that if there are any external factors that can influence the demand, it would be beneficial to use when making predictions.

Lei et al. [31] focus mainly on weighted graphs. This approach is named Graph Convolution Network - Generative Adversarial Network (GCN-GAN) and uses GCN and GAN to predict the weighed links. First, GCN is used to capture the topological structure of each snapshot, similar to Zhao's and Ge's approaches. Then by the use of an LSTM, the evolvement of the dynamic weighted network is captured. Finally, GAN is used to generate high-quality weighted links. In short, GAN is used to deal with the sparsity and wide-value range of the weights of the edges.

The above-defined approaches have one significant advantage compared to the approaches defined in Section 3.1. This advantage is that these methods are trained to capture the non-linear transformation of the dynamic network over time. This increases the complexity of these models. A more complex model does not always result in better performance, as is shown by, [2], where the performance of the ConvLSTM is comparable to the performance of the Markov chain, and only sightly outperforms the Markov chain on a small subset of datasets. This highlights that it is an appropriate choice when making predictions to start with the less complex models. If these models do not perform well, and therefore cannot learn the characteristics of the evolvement of the network, more complex models are needed. When this is the case, there are several options. As most approaches are based on convolution operation, the ConvLSTM is a good first option. This approach is the most basic one of the approaches mentioned above. If this does not increase the performance, other optimized variants of this algorithm, for example, the GCN, can be implemented. In Section 6.2, it will be tested which model is most suitable for predicting the weight of the links in the demand network.

3.3. Bipartite Link Prediction Methods

Because the problem in this thesis is modeled as a bipartite network, the literature was further explored to find methods that mainly focus on bipartite networks. It is interesting to research if the fundamentals of the approaches, discussed in the previous sections, differ from the approaches that focus on bipartite networks. A very diverse set of approaches was found in the literature, mainly because these models try to solve different problems.

The first category of methods is the projection-based approach. The approaches within this category create a unipartite network from the bipartite network. This created network is called the projected graph. [32], [33], and [34] all proposed a method that tries to solve the link prediction problem by projecting the bipartite network into a unipartite network. Based on this unipartite graph, the models try to predict which links will occur in the future which did not exist in the past. Predicting new links in a network is extremely popular and essential in the E-commerce field. However, in this project predicting new links is inapplicable. Therefore the projection-based approach is not a suitable method for solving the demand prediction problem.

Kart et al. [35] and Chai et al. [36] both applied (supervised) machine learning techniques to predict the structure of the network. First, low dimensional input features were created. Kart et al. created input features using neighbor-based metrics (e.g., Jaccard Coefficient) and path-based metrics (e.g., random walk), and several machine learning techniques (e.g., Naive Bayes, Support Vector Machines, Random Forest) were used to make the predictions. Chai et al. also use a special type of random walk, and together with the skip-gram model, low dimensional input features were created. Again several machine learning algorithms were used to predict the structure of the network. Like the projection-based approaches, these approaches only focus on predicting new links in the network and are not used to predict the weights of existing edges. Islek et al. [37] proposed a model that can predict the weight of a link. In their paper, the demand for warehouses is forecasted and comparable to this project's problem. The warehouses (left side nodes of the bipartite network) were clustered based on their selling behavior. Then, using the moving average values of the demand, warehouse-related attributes, and product-related attributes, a Bayesian Network model was constructed. Finally, using the Bayesian Network, predictions were made.

Jin et al. [38] highlights a very important problem when GCNs are used to predict future bipartite network structures. GCNs are initially designed for unipartite networks. When GCN is applied to a bipartite network, it needs to combine two different types of information because of the two different types of nodes in the bipartite network. It also uses the same weight matrix for the filter operation, and Jin proved that this does not work correctly. An improved version, the Bipartite Graph Convolutional Network (BIGCN), is proposed to solve the problem. The key point of the proposed solution is to use a separate convolution operation for the two types of nodes. However, this approach tries to predict a binary label for the weights of the edges in the network. Besides, it also does not take the temporal aspect into account.

Mutinda et al. [39] combine two techniques to predict the future structure of the network. This approach first applies (nonnegative) matrix factorization to extract the latent features of the historical graphs. Based on extracted latent features, multiple forecasting algorithms (e.g., Holt-Winters, VAR, LSTM) are used to predict the network's structure. As the objective of this paper was to predict hidden links, in other words, links that do not appear in historical graphs, the algorithms used in their paper are not optimized for the problem this thesis wants to solve.

This section highlights that the temporal link prediction in bipartite networks mainly focuses on predicting new links. This is because the bipartite networks that are often researched are user-item networks. In these networks, it is of high value knowing when a user will connect to an item. Another interesting observation is that applying some of the methods discussed in the previous section on bipartite networks will probably not result in the expected prediction accuracy as proved by Jin et al.. Therefore, before implementing the models, it is important to know which data will be used and how it is used.

To conclude, in this chapter several types of algorithms have been discussed. The complexity of these approaches differs significantly. By using the less complex models first, it can be seen whether these less complex models can capture the network evolution. The next step would be implementing the more complex models and compare the performances of these models. Because the performance of the models depends highly on the network structure, it is impossible to conclude which models are most suitable beforehand. However, it is possible to analyze the predictability of the dataset, and the results of this will be discussed in Section 4.3.

4 Data Analysis

To be able to analyze the data, the raw data received from the company was cleaned up. How the data is cleaned will be explained in this chapter. Also, an explanation is given on what part of the data is used to make the predictions. Lastly, the results of the data analysis will be presented. Based on the results, several design choices are made.

4.1. Data Preprocessing

Data-driven models are proven to be very effective and powerful and have several advantages over model-driven models. As stated by Montàns et al. [40] data-driven models have the advantage of testing correlations between different variables and observations, learning unforeseen patterns in nature, and allowing us to discover new scientific laws or performing predictions without the availability of such laws. However, the well-known paradigm "garbage in, garbage out" already emphasizes the importance of the quality of the input data. In this section, an overview is given of the steps taken to create high-quality data.

4.1.1. Generating Input Data

The data elaborated in Section 2.2 contains three data types: Integers, Floats and Strings. Because the data is stored in a comma-separated values (CSV) file, it was imported with ease. No complex prepossessing steps were needed. Some of the basic prepossessing steps taken are rounding of Floats, converting Floats to Integers and vice versa, converting the TimeStamps from data type String to Python's Datetime64 format, adding translations of Dutch words, as well as joining and merging multiple datasets.

From the unit log, we can compute the amount of time a unit stays in a specific state using a relatively simple SQL query. In Table 4.1, a small snapshot of the data is shown, where only the fields used in this project are shown:

Table 4.1: Snapshot of the data containing the amount of time a unit stays in a specific state.

BusinessUnitID	UnitTypeID	UnitGroupID	UnitGroupEnglish	UnitID	UnitVolume	UnitStatusID	UnitStatusEnglish	Start_date	End_date	Number_of_days
6	690	7	Ground floor	14104	1.0	1	In use	2006-08-22	2012-01-16	1973
6	690	7	Ground floor	14104	1.0	0	Free	2012-01-17	2012-01-30	13
6	690	7	Ground floor	14104	1.0	5	Reserved	2012-01-31	2012-02-01	1
6	690	7	Ground floor	14104	1.0	0	Free	2012-02-02	2012-03-09	36
..	..	..	..	..	..	..	..	..	..	..
54	765	14	Fifth floor	156352	1.0	0	Free	2021-01-22	2021-03-01	38

From this dataset, it is possible to compute for each UnitLTG the exact number of rented units per day. It is also possible to compute the occupancy rate per week by taking a seven-day average of occupancy rates. However, data on a lower timeframe provides more resolution, and hence it contains more information that can be used to predict the future. During the implementation and evaluation, it will be analyzed whether using daily data is an appropriate choice.

In order to compute the occupancy rate per day, first, the dataset is filtered, such that it only contains rows where the UnitStatusID is equal to 1 (in use). It is important to note that this project only focuses on analyzing the demand, and therefore, the dataset could be filtered as described above. Analyzing other patterns, for example, the amount of time it takes before a customer decides to rent a unit after reserving a unit is outside this project's scope due to the limited time and the in-depth analysis of other data. Although, this information can be very valuable for the company since there will likely be a correlation in customers' decision-making process. For this example, rows with UnitStatusID 5 (reserved) are also necessary for possible future use.

The matrix, R, stores the total number of rented units for each UnitLTG per day. R is an $m \times n$ matrix, where $m = 2397$ and represents the number of unique UnitLTGs, and $n = 7971$ and denotes the number of days the company exists. Because it is exactly known when a unit is rented, presented by the Start_date and End_date column shown in Table 4.1, matrix R can easily be updated. Briefly summarized, for each row in the table, the correct row index (UnitLTG) in the matrix R must be found, and for this row, the cells between columns Start_date to End_date need to be incremented by one. The pseudocode for creating matrix R can be seen in Algorithm 1.

```
Algorithm 1: Creating the matrix containing per day the number of rented units per UnitLTG

Result: Matrix containing per day the number of rented units per UnitLTG
total_rented_units = empty $m \times n$ matrix initialized with zeros
for cur_row in unit_log do
    row_index_matrix = find row index of total_rented_units for the UnitLTG of cur_row
    for i in range(Start_date, End_date) do
        total_rented_units [row_index_matrix, i] += 1
    end
end
return total_rented_units
```

The result of the algorithm defined above contains for each time step (day) the total number of rented units for each UnitLTG. For illustration purposes, a small snapshot of the matrix is shown in Table 4.2.

Table 4.2: Matrix containing for each day the total number of rented units for all UnitLTGs.

			05-05-1999	06-05-1999	07-05-1999	..	27-02-2021	28-02-2021
Leiderdorp	1.0m^3	Ground floor	0	0	0	..	21	21
Leiderdorp	1.0m^3	Third floor	0	0	0	..	27	27
Leiderdorp	1.5m^3	Ground floor	0	0	0	..	8	8
Leiderdorp	1.5m^3	Second floor	0	0	0	..	43	43
Leiderdorp	1.5m^3	Third floor	0	0	0	..	12	12
..	..	..	..	..	..	..	..	..
Oegstgeest	24m^3	Basement	0	0	0	..	0	0

In Section 4.1.2, it will be explained which part of the data is used in this project, and an explanation is given why not the entire dataset is used.

4.1.2. Data Selection

As already highlighted in Section 2.2, the unit log contains some inconsistencies. Therefore, a subset of the data is taken such that it only contains consistent data. This section will explain what a consistent dataset means and how to select it.

First of all, to analyze patterns in demand, a certain amount of data needs to be available. It is not possible to analyze yearly patterns when only a few months of data is available. Thus, the first selection criteria are the number of years of data available. In this project, the threshold is set to four years, which means that n = 1460. Locations that have existed for less than four years will not be taken into account. The minimum is set to four years because the amount of information would be sufficient to observe whether the demand contains any patterns which can be used for predictions. This number can be slightly increased or decreased when the performance of the model is not as expected.

Another selection criteria are the set of UnitLTGs and UnitLTs used. In this project, the set of UnitLTGs and UnitLTs includes all locations where no changes have been applied, either in the number of units (e.g., by expanding) or changes in volume. The number of units of a UnitLTG or UnitLT could, for example, increase when several smaller units replace a large unit. This could be because the demand for small units is very high and the demand for large units very low. Because this could not be known beforehand, it happens that a large unit is split up into several smaller units. However, this change does have a significant impact on the demand because more units become available, resulting in a drop in the demand. Therefore, only the UnitLTGs and UnitLTs are selected where the number of units is constant over time. Based on these conditions, the demand, D at time step t, can be calculated using Equation 2.1.

As can be seen in Equation 2.1, the demand can be calculated by dividing $N_{occupied}$ at time t by N_{total}. Thus, each row in in matrix R shown in Table 4.2, needs to be divided by N_{total}, which is the number of unique units in the UnitLTG. It is important to note that it needs to be checked whether all units of the UnitLTG exist for at least four years. If that is the case, then it is used in this project. The final matrix, shown in Table 4.3 contains only UnitLTGs, of which the number of units does not increase over time. As can be seen, the rows are represented by the UnitLTGs, and the columns relate to the specific day. The cell values represent the percentage of rented units and are always between 0 and 1. A similar matrix can be created for the UnitLTs, but this matrix is not shown to keep a clear overview.

Table 4.3: Matrix containing for each day the demand for all UnitLTGs.

			27-02-2017	28-02-2017	..	27-02-2021	28-02-2021
Leiderdorp	1.0m^3	Ground floor	0.935	0.915	..	1.000	1.000
Leiderdorp	1.0m^3	Third floor	0.900	0.920	..	1.000	1.000
Leiderdorp	1.5m^3	Ground floor	0.850	0.910	..	0.800	0.800
Leiderdorp	1.5m^3	Second floor	0.750	0.800	..	0.935	0.935
Leiderdorp	1.5m^3	Third floor	0.935	0.935	..	0.857	0.857
..	..	..	..	..	..	..	..
Oegstgeest	24m^3	Basement	0.935	0.935	..	1.000	1.000

As mentioned earlier, a UnitLTG is a set of units with the same location, volume, and floor. The total number of unique UnitLTGs with a least one unit, defined as U$_{ltg}$ = 2397, while the number of unique UnitLTs, defined as U$_{lt}$ = 1092. However, after selecting the UnitLTGs, that satisfy the selection criteria, defined as U$^*_{ltg}$, the number of unique sets is reduced by 68.5%. This means that in this project, the number of unique UnitLTGs, U$^*_{ltg}$ = 756. No difficulties were encountered while creating a temporal weighted bipartite graph from the preprocessed dataset. The created bipartite graph contains 18 nodes (locations) on the left side of the graph and 196 nodes (set of units with the same volume and floor) on the right-hand side of the graph, and the network contains 756 edges. The same process is repeated for the set of UnitLTs, where the floor at which the units are located is not taken into account. As defined above the size of U$_{lt}$ = 1092, after selecting only UnitLTs where the number of units is stable over time, defined as U$^*_{lt}$, the number of UnitLTs is dropped by 67.2 %, resulting in U$^*_{lt}$ = 359. Again a bipartite graph could be created, resulting in 18 nodes (locations) on the left side of the graph and 48 nodes (set of units with the same volume) on the right-hand side of the graph and in total 359 edges. In the following section, an analysis will be done to see if the floor on which the units are located influences the occupancy rate of the units. If this is the case, the floor needs to be taken into account, and the nodes at the right-hand side of the bipartite network need to be a combination of the volume and floor. Otherwise, the right-hand side nodes can be represented by the volume only.

4.2. Unit Analysis

The previous section explains which part of the data will be used for the predictions. In this section, an overview is given of the number of units per location. This section, therefore, illustrates the impact of the data selection. Besides that, it will give an overview of the most frequently used units. A heatmap is an effective tool to sketch a high-level overview of the correlation between two variables. The two variables used are the location and the unit volume, and thus the floor on which the units are located is not taken into account. The color intensity is used to highlight the number of occurrences of the units for the different locations. Using a heatmap, it can easily be shown which units are most frequently used.

In Figure 4.1, the heatmap is shown. The x-axis represents the unit volume, while the y-axis the different locations. The higher the number of units, the darker the cell value of the heatmap. It is worth mentioning that not every unit volume is shown on the x-axis. The units that are not included are mainly special types of units which are not frequently used. The figure shows that the smaller units are used more and unit volumes 3m^3 to 12m^3 are the most frequently used across the different locations. Interestingly, Sittard does not contain any of the above-defined unit volumes. This is a result of the data selection defined in the previous section. It is proven that the number of units for these specific unit volumes at Sittard increases over time and is therefore not included in the data analysis. This could, for example, be a result of an expansion of the size of the storage building, which makes it possible to increase the units that are high in demand.

Also, the number of floors increases over time until their maximum is reached. This works as follows: when a new location is build a mixture of units is built on the ground floor. When an occupancy rate of 80% is reached, a new layer is built on the existing floor. This process is repeated until the maximum number of floors is reached. This process is used because the demand for the volumes depends on the geographic location of the storage buildings. Based on the demand of the unit volumes at the existing floors, the optimal mixture of unit volumes of the next floor can be predicted.

If time permits, the data selection could be changed to take the above-defined growth of the number of units into account. An interesting research problem that arises from this change is analyzing the optimal mixture of units at a location. Moreover, especially analyzing the following problem: "What is the optimal mixture of unit volumes at a new layer based on the existing layers?". Looking from a bipartite network perspective, this could be analyzed by adding new nodes to the right-hand side of the bipartite network and predicting the demand for these new links in the network. Because this problem is not comparable to predicting the demand of the existing units, it will only be addressed when time permits. Note that this problem can be compared to the cold start problem in recommendation systems [41].

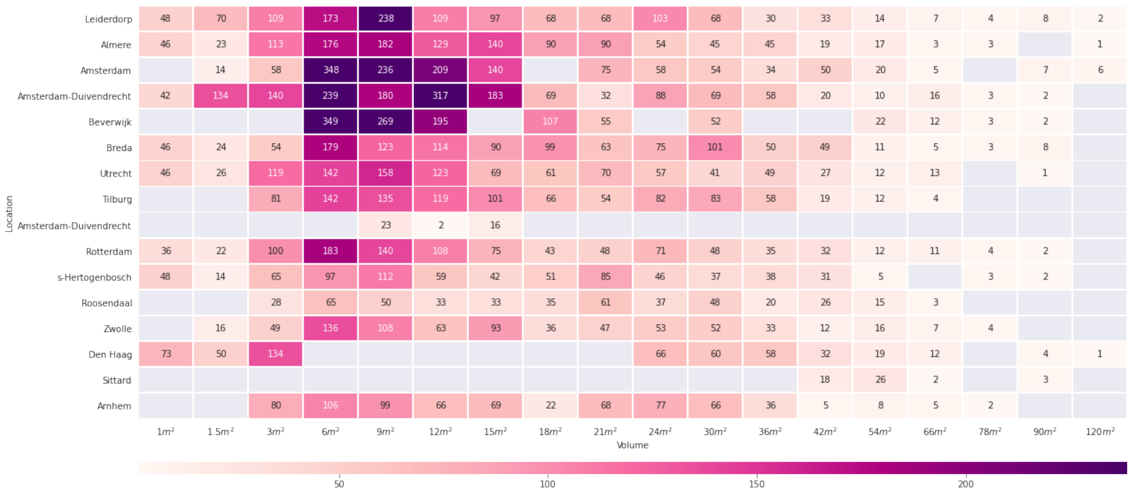


Figure 4.1: Heatmap containing the number of units per locations.

4.3. Demand Analysis

In this section, the demand is analyzed. First, an overview of the distribution of the occupancy rates is given, and it is analyzed which variables influence this distribution. Also, statistical methods are used to detect any (periodic) patterns in the temporal change of the demand.

4.3.1. Distribution

To predict the weights of the links in the network, it is crucial to understand the distribution of the weights used to make the predictions. This is important because it could already give insights into the range of the predicted value. Besides that, it is required to select appropriate metrics to evaluate the model's performance. Lastly, the distribution could be used to compare locations and types of units.

4.3.1.1 Volumes

Because the demand ratio of a single link changes over time, the demand values of a single link in the bipartite network can be represented as a time series graph. Therefore, several statistical methods can be used to analyze these time series. In Figure 4.2, an example of such a time series graph is shown. This graph shows how the occupancy rate of the units with $V = 6\text{m}^3$ that are located on the first floor in Leiderdorp changes over time. From this graph, it can be seen that $D > 0.87$ in the selected timeframe and does reach its maximum occupancy several times in the last four years.

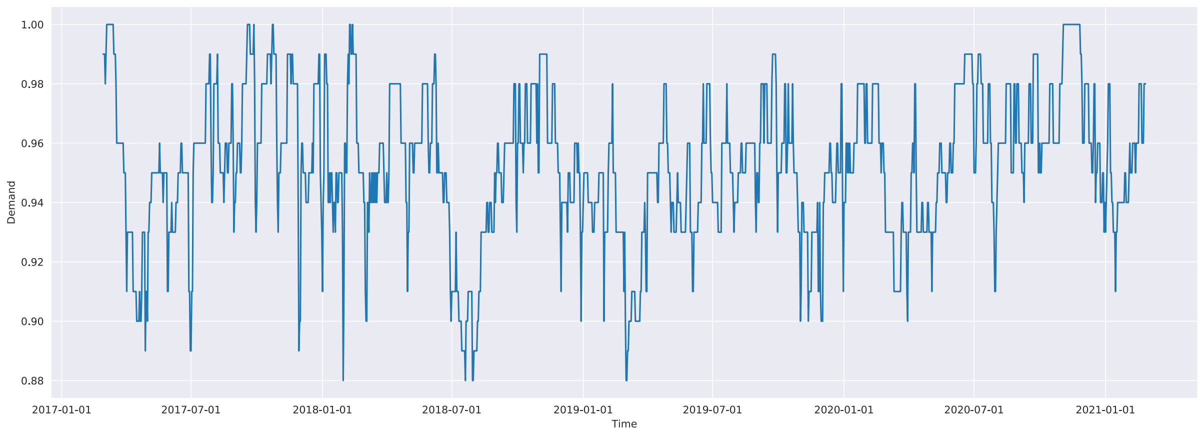


Figure 4.2: Example of the temporal change of the demand for a randomly selected type of unit and location.

The probability density function (PDF) is used to estimate the distributions, to compare the different distributions across the different locations and types of units. In total, three different distributions will be analyzed. First, the demand distribution will be analyzed from a high-level point of view, meaning the demand of all units of equal volume is aggregated. Then, the demand distribution of the UnitLTs will be compared to the distribution of the demand of the UnitLTGs. How this is done will be explained in more detail later in this section.

To plot the PDF the data is divided into discrete bins, then the number of data points that fall in each bin is counted, and finally the results are visualized. The width of the bins could be determined in several ways. A frequently used approach is to determine the bin width is defining the number of bins beforehand and calculating the width of the bins with Equation 4.1.

$$bin\_width = \frac{max\_value - min\_value}{number\_of\_bins} \tag{4.1}$$

Because the difference between the maximum and minimum values of the different time series differ significantly, the bin width of the different plots was not comparable, making it hard to compare the plots with each other. Therefore, the bin width was set to a fixed value by trial and error.

The kernel density estimation is used to estimate the probability distribution of the dataset. The estimated probability function at a particular point x, $f^*(x)$, can be computed using Equation 4.2.

$$f^*(x) = \sum_{data\_points} K\left(\frac{x - data\_point}{bandwidth}\right) \tag{4.2}$$

As can be seen in Equation 4.2, two variables need to be set beforehand, the kernel function K and the bandwidth. The default values for these two variables are used because the estimated probability distribution models the actual distribution relatively good, which can be seen later in the plots. The default kernel function used by the kernel density estimation is the Gaussian kernel and the bandwidth is set using Scott's rule of thumb [42] and is calculated using Equation 4.3,

$$bandwidth = 3.5 * \sigma_{data} * \sqrt[3]{n} \tag{4.3}$$

where σ_{data} represents the standard deviation of all data points and n the total number of data points.

In Figure 4.3, the distribution of eight different unit volumes is plotted, respectively 1.0m^3, 1.5m^3, 3.0m^3, 6.0m^3, 9.0m^3, 12.0m^3, 15.0m^3, and 18.0m^3. In these plots, the demand ratios of all locations are combined to sketch a high-level overview of the demand distribution. This high-level overview can be used to see whether the unit volume impacts the demand. The PDF is plotted against the demand ratio. The blue line represents the estimated probability density function. The title indicates which data is used for the plots. From the plots, it can be concluded that the demand distribution depends on the unit volume. Some units have comparable distributions, for example, unit volumes 15m^3 and 18m^3. The range of these distributions is approximately between 0.6 and 1.0. Both distributions are highly centered between 0.8 and 0.95. However, if these distributions are compared with the distribution of smaller units, for example, unit volumes 1m^3 or 3m^3, it can be seen that the difference between the distributions is significant. Based on these results, it can be concluded that the volume is a variable that influences the distribution of the demand. Therefore, it needs to be taken into account when making predictions.

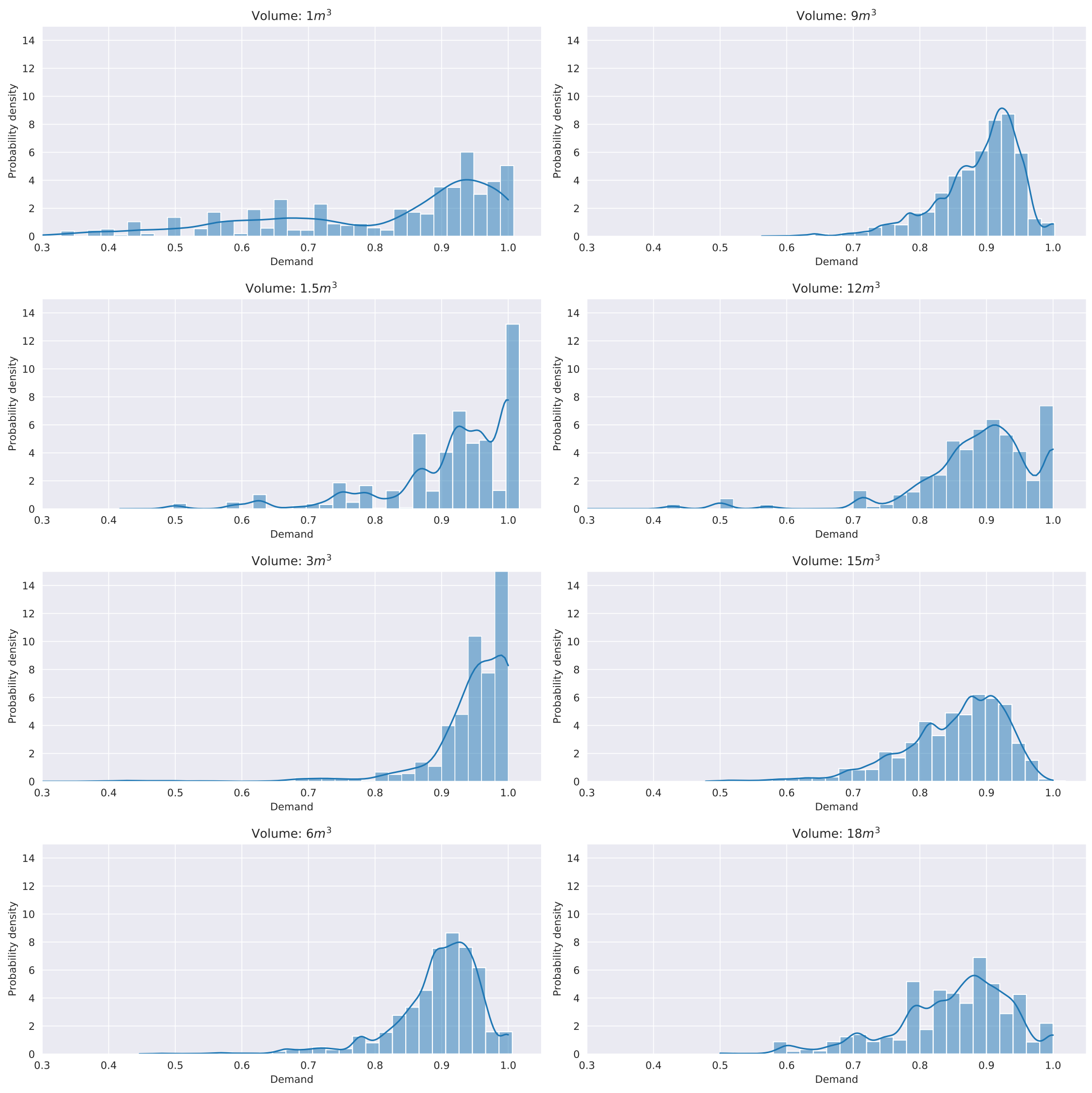


Figure 4.3: Estimated probability density function for the combined demand for several unit volumes.

4.3.1.2 Floors

Previously, it was proven that the unit volume impacts the distribution of the demand. Another feature that could impact the demand distribution is the floor at which the units are located. Similarly, as the approach used to analyze the volume's impact, the impact of the floor is analyzed. The results have been proven to be very comparable to the analysis of volume, seen in Section 4.3.1.1. The interested reader is referred to Appendix B.1.1 for more details.

4.3.1.3 Locations

The last variable that could impact the demand is the location where the units are rented. Likely, the demand for the locations located in the north of the Netherlands is not comparable to the demand for the storage units at sites in the Randstad, the center of the Netherlands. By comparing the demand distribution for the same unit type across different locations, it is verified that the location impacts the demand distribution. Again, for the distributions of the demand ratio for the specific storage locations, one can read Appendix B.1.2.

The results of this section have proven that the location, volume, and floor all influence the distribution of the demand. Therefore, creating a bipartite network, where the left-hand-side nodes represent the locations and the right-hand-side nodes a combination of the volume and floor, is a suitable way to model this problem. In Table 4.4, several graph metrics are shown to summarize the structure of the bipartite network. As can be seen, the graph consists of 214 nodes and 756 edges. The degree of a node is defined as the number of connections a node has to other nodes in the network. In this network, each node is, on average, connected to 7.07 nodes. However, the standard deviation is 13.79, which means that the average degree varies widely. This is also expressed by the difference between the minimum and maximum degree. Lastly, the link density proves that the network is extremely sparse because it only contains 3.0% of the total number of links.

Table 4.4: Statistics of the bipartite network.

Graph metric	Value
Total number of nodes	214
Number of left-side nodes	18
Number of right-side nodes	196
Total number of edges	756
Average degree	7.07
Standard deviation degree	13.79
Minimum degree	1
Maximum degree	89
Link density	0.03

Because the problem is modeled as a network, the correlation between locations or units can be used to make predictions. Modeling the problem without extracting the cross-site information, valuable information might be lost, as the situation is observed for each unique combination of location, unit volume and its floor. Later in this report, it will be analyzed whether a high correlation exists across the locations and unit types.

4.3.2. Autocorrelation

When making predictions, it is important to know how future data points correlate with historical data points. If there is no correlation, the model can be called a random process. Hence it is impossible to make accurate predictions. Because the demand can be represented as a time series graph, statistical tools can be used to analyze how the demand changes over time. The autocorrelation function (ACF) is a well-known approach that can be used to identify seasonality and trend in time series data. The ACF computes the correlation of a time series with a shifted version of itself. If the shifted time series behaves similarly to the original time series, the correlation is high. The autocorrelation will be computed for several cases. First, the average autocorrelation of all locations for a single unit volume will be computed. Then, the average autocorrelation for all units and locations will be analyzed.

The autocorrelation is computed using the Pearson correlation coefficient and is computed only on the overlapping part of the time series. The correlation coefficient between two vectors b and c can be computed using Equation 4.4,

$$P = \frac{\sum_{i=0}^{n-1}(b_i - \bar{b})(c_i - \bar{c})}{\sqrt{\sum_{i=0}^{n-1}(b_i - \bar{b})^2 \sum_{i=0}^{n-1}(c_i - \bar{c})^2}} \tag{4.4}$$

where $\bar{b}$ denotes the mean of b and $\bar{c}$ represents the mean of c and n the length of the vectors.

As mentioned above, the autocorrelation is only computed on the overlapping part of the time series. More formally, let X ($\{X_0, X_1, X_2, .., X_n\}$) be the time series with n being the length of the time series X. The number of steps the time series is shifted is often called lag, defined by k. When $k = 0$, the Pearson correlation will be computed between two identical time series and will therefore always be 1. When $k = 1$, Equation 4.4 is computed with $b = \{X_0, X_1, X_2, .., X_{T-1}\}$ and $c = \{X_1, X_2, X_3, .., X_T\}$. So, in general when $k = h$, Equation 4.4 is computed with with $b = \{X_0, X_1, X_2, .., X_{T-h}\}$ and $c = \{X_h, X_{h+1}, X_{h+2}, .., X_h\}$. The autocorrelation is computed for $1 \leq k \leq 365$. The maximum number of lags is set to 365, which means that the time series is shifted by a maximum of one year.

In Figure 4.4, the average ACF, defined as A, has been computed and plotted for $V = 6\text{m}^3$. To plot the average autocorrelation, the autocorrelation for the selected volume per city must be computed. Then based on these computed autocorrelations, the average autocorrelation per lag is computed as well as their standard deviation. Analysis over all unit volumes has given the insight that there are no significant differences, which is why the correlation output for only a single unit volume is presented. The dark blue line represents the average autocorrelation, while the light blue shaded area shows the standard deviation. It can be seen that the average autocorrelation has a very high standard deviation (shaded area). This result highlights that the location where the units are rented impacts how the occupancy rate changes over time. This result will be discussed in more detail in the next section.

Besides the high standard deviation, two main observations will be discussed. First of all, the autocorrelations for the different volumes all contain a similar pattern, namely, for $0 \leq k \leq 35$, the autocorrelation decreases significantly towards $A \approx 0.5$. For $k > 35$, the autocorrelation slowly decreases and never increases much. Based on this result, it can be concluded that when making predictions, only a limited number of data points can, or should be, used because of the high correlation at a small number of lags and an only decreasing autocorrelation function. Thus when more (less correlated) historical data is used, it will probably negatively impact the accuracy of the predictions. Furthermore, it can be seen that there are no large peaks or lows, and this indicates that the data does not contain significant periodic patterns.

The slower the average autocorrelation function decreases in the first lags (across all UnitLTG), the higher the model's predictability will be. To find out what the average ACF is for a small number of lags, the average ACF is computed for each UnitLTG. The average ACF of all UnitLTGs is plotted against the number of lags as shown in Figure 4.5. The number of lags is limited to 31 because, similar to the result of Figure 4.4 the autocorrelation does not contain significant peaks or lows and is an only decreasing function. Figure 4.5 emphasizes that, on average, there is a very high correlation between consecutive data points because of the high autocorrelation at the first few lags. It can be seen that when $k = 7$, $A > 0.8$, which can therefore be considered as significant. It can also be noticed that the standard deviation at the first few lags is smaller than for a larger number of lags. This result indicates that the variance of the autocorrelation increases when the number of lags increases. The same process is repeated for the UnitLTs because it produces a similar result it is not shown in this report.

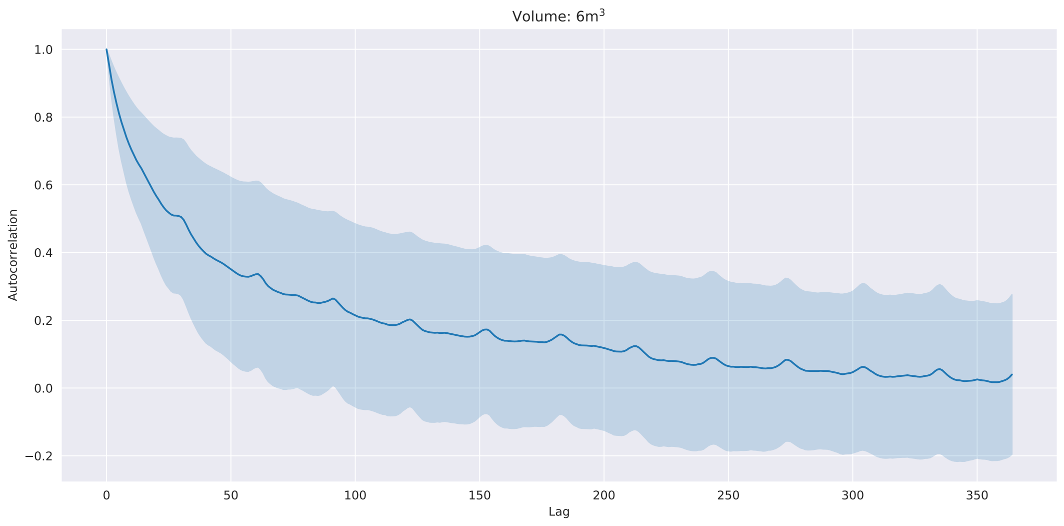


Figure 4.4: The autocorrelation output for all UnitLTGs for $V = 6\,\text{m}^3$. The blue line represents the mean ACF output, while the shaded area denotes the standard deviation across the UnitLTGs.

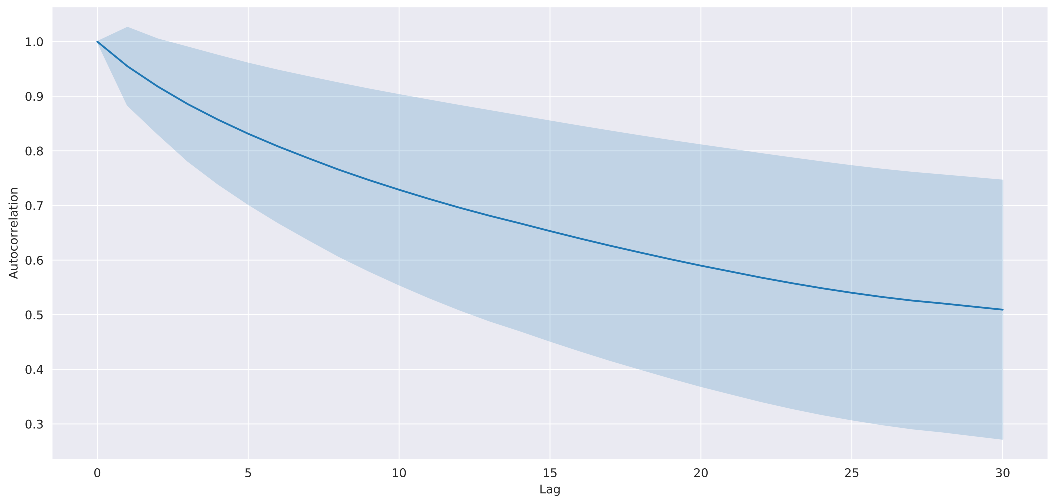


Figure 4.5: The autocorrelation output for all UnitLTGs. The blue line represents the mean ACF output, while the shaded area denotes the standard deviation across the UnitLTGs. Note that the x-axis only contains data of a single month instead of a year

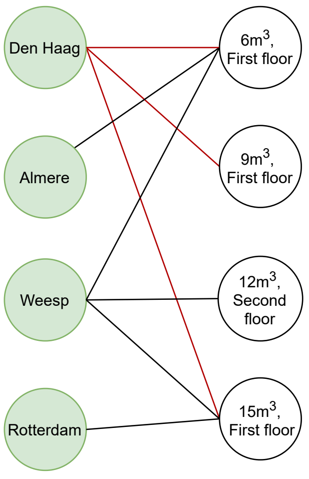
The results discussed in this section emphasize that it is possible to predict the occupancy rate of future time steps because of the high correlation at the first few lags in the time series data. Also, it is proven that the location heavily impacts how the occupancy rate changes over time. Besides that, it is shown that when making the predictions, only a few historical data need to be available because the autocorrelation only decreases over time and does not contain any periodic patterns. The cross-correlation will be used to statistically compare how different time series changes over time in the next section.

4.3.3. Cross-correlation

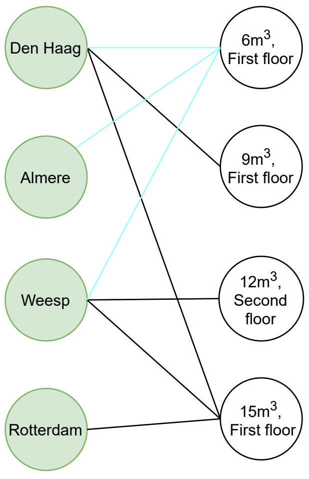
In this section, the temporal change of the demand of the different time series will be compared. The cross-correlation function (CCF) is a statistical tool to find out how well two time series match up with each other. And could therefore be used to find out how the temporal change of the occupancy rate of one type of unit is related to the temporal change of another type of unit.

Equation 4.4 can be used to compute the cross-correlation between two different time series. Given two time series X ({X_0, X_1, X_2, .., X_n }) and Y ({Y_0, Y_1, Y_2, .., Y_n }) with n being the length of the time series. The cross-correlation for $k = h$ can be computed using Equation 4.4 with $b = \{X_0, X_1, X_2, .., X_{n-h}\}$ and $c = \{Y_h, Y_{h+1}, Y_{h+2}, .., Y_n\}$ for $0 \leq h \leq 365$.

Because the problem is modeled as a network, the correlation between links can be taken into account when making predictions. Therefore, the CCF is used to analyze the correlation between the links in the network. The CCF is computed between two meaningful sets of adjacent edges of the bipartite network. The first set of edges have the same location/source node (left side of the bipartite network) but a different unit type/destination node (right side of the bipartite network). An example of such a set is given in Figure 4.6a, where the red colored edges represent the set of edges that have the same location but a different unit type. The other set of edges have a different location/source node (left side of the network) but the same unit type/destination node (right side of the network). An example for this set of edges is shown in Figure 4.6b, where the set of edges have an equal unit volume and floor but a different location and are represented by the light-blue colored edges.



(a) Example of set of edges with same location and different unit volume and floor.



(b) Example of set of edges with a different location but equal unit volume and floor.

Figure 4.6: Example of a set of edges used to compute the average CCF.

The average cross-correlation, defined as Z, plotted in the next figures, is computed in the following way. First, the time series are selected based on one of the scenarios defined above. Assume there are m outgoing edges from node n, then in total m times m - 1 cross-correlations are computed because it does matter which time series is lagged. Then, based on the m times m - 1 computed cross-correlations, the average cross-correlation is computed for each lag. This process is repeated for each node in the network that satisfies the selection criteria. Finally, based on the computed cross-correlations, the average cross-correlation per lag with their standard deviation is computed.

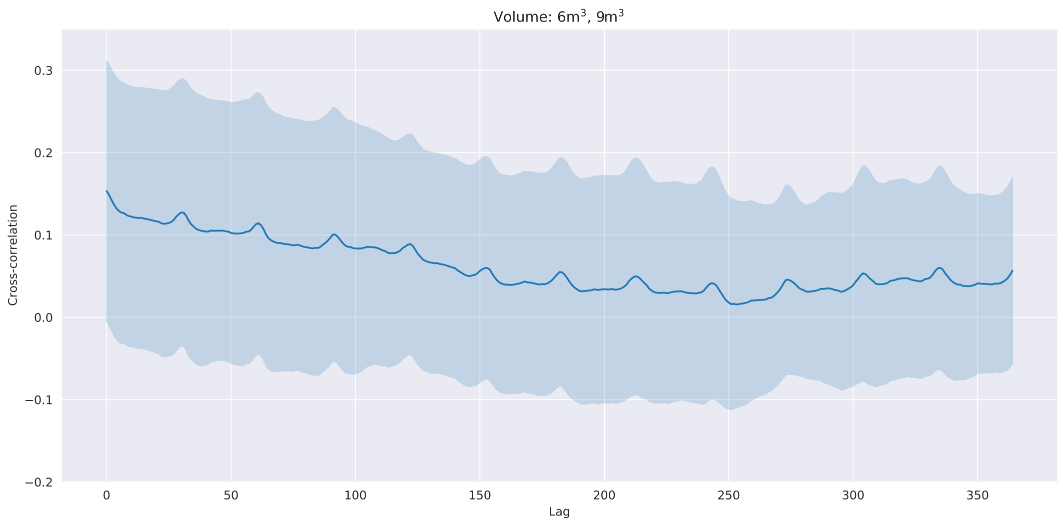


Figure 4.7: Average cross-correlation between the demand of unit volume 6m^3 and 9m^3.

Figure 4.7 shows the cross-correlation output between $V_1 = 6\text{m}^3$ and $V_2 = 9\text{m}^3$ for all UnitLTGs. The range of the y-axis is between -0.2 and 0.3. From the plot, it can be seen that the average CCF is relatively small and $0.0 \leq Z \leq 0.2$ for all k. Because the cross-correlation output is close to 0, it can be concluded that there is a negligible correlation between the time series. Just like previous plots, the standard deviation (shaded area) is large. This indicates that the cross-correlation between the demand of unit volume V_1 and V_2 heavily depends on the location and does not contain similar patterns. It could be the case that other unit volumes are more correlated and therefore have a higher cross-correlation. The cross-correlation between every pair of UnitLTs and UnitLTGs is computed to see if this statement is correct. The cross-correlation between the pairs of UnitLTs is only computed if the location of the UnitLTs is the same. In other words, the units with a different volume but equal geographic location are compared. The same holds for the UnitLTGs, however, these units are compared with similar geographical location and floor. It is worth mentioning that the cross-correlation at lag zero is not equal to 1, except when two time series behave similarly.

In Figure 4.8, the average cross-correlation between all possible pairs of UnitLTGs that are located at the same location and floor is computed and plotted. As can be seen, the average cross-correlation is low for every lag and only decreases when the number of lags increases. It is noticeable that the average cross-correlation and standard deviation at the first few lags are significantly larger than for a larger number of lags. This result indicates that time series are more correlated when comparing time series when k is small. However, the value of the average cross-correlation is still not significant. The high standard deviation for $0 \leq k \leq 20$ could result from a subset of time series that have a high correlation. In general, it can be concluded that that the temporal change of the demand of one unit type does not provide any useful information about the temporal change of a different unit type. This is an important finding because it emphasizes that the temporal change of the demand of the units is non-trivial and does not contain any significant patterns that make the predictions more simple. This highlights that predicting the demand for certain steps ahead is an interesting problem to analyze because the demand changes uniquely over time, and the different time series do not contain significant similarities. The same process is repeated for the UnitLTs. The results were almost identical, and therefore, not shown in this thesis.

In the previous section, it was already mentioned that the site impacts how demand changes over time. This was shown by the high standard deviation in, for example, Figure 4.4 that shows the average autocorrelation of all locations for a single unit volume and similarly Figure 4.5 showing the average cross-correlation of all locations for two unit volumes. To find out if this is the case for all locations, the cross-correlation between all possible pairs of UnitLTGs will be computed. However, now the UnitLTGs are compared if they have the same unit volume and are located on the same floor but have a different geographical location (the light-blue lines represented by Figure 4.6b). Based on the line graph containing the average cross-correlation shown in Figure 4.9, it can be seen that the temporal change of the units with the same volume and floor but a different

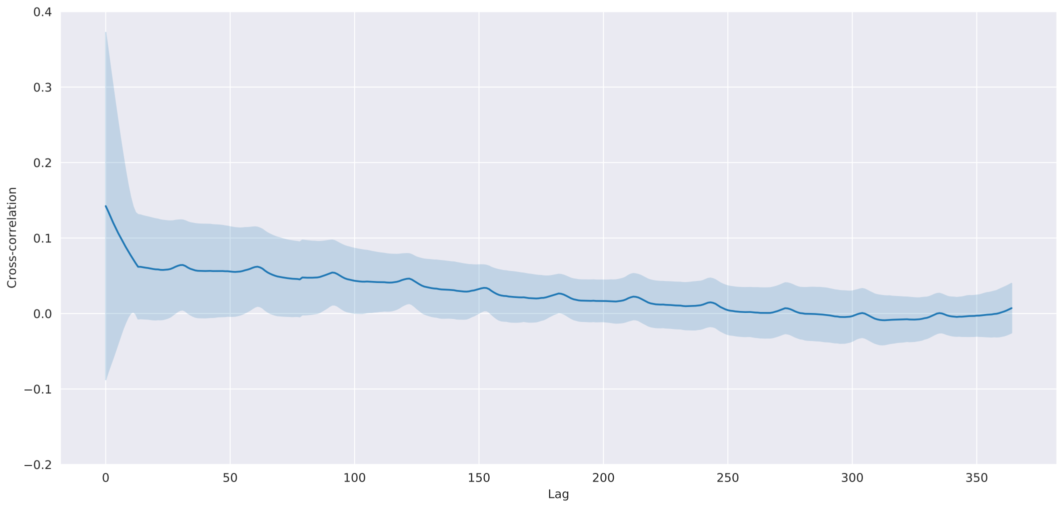


Figure 4.8: Average cross-correlation for all possible pairs of UnitLTGs that are located at the same location and floor.

location are completely uncorrelated. Just like the previous plot, the average cross-correlation is close to 0 for every lag. The standard deviation is very large for each lag, meaning that the variability of the values is high. The same plot could be created for the UnitLTs but again the plot is similar to Figure 4.9 and is therefore not included in this report.

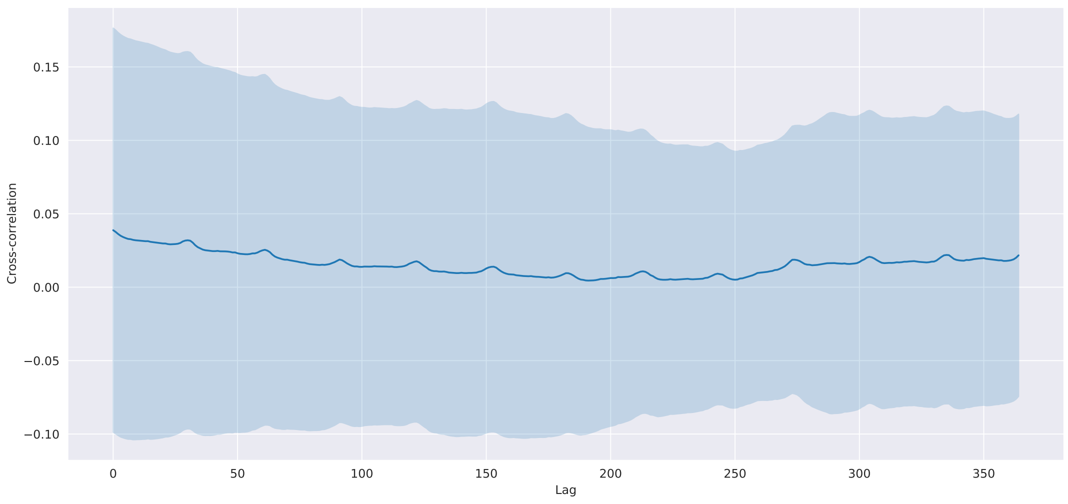


Figure 4.9: Average cross-correlation between units with same volume and floor but a different city.

The results in this section emphasize that the temporal change of the demand of the different units has no significant correlation in general terms. Thus, it can be concluded that the temporal change of the different units is unique and does not contain similar patterns. Therefore, using data of different time series, in other words, using the temporal change of the demand of different units, will probably not help when predicting the demand of other units. It is also shown that temporal change in demand of equal unit types but located at different cities does not contain any correlation.

4.3.4. Summary

Reflecting on research subquestion A1 (see Section 1.3), "Does the data consist of any patterns, if so, which patterns indicate the predictability of the network?" a few insights have been gained. As shown in the previous sections, the data analysis has shown that several patterns exist. First of all, it is proven that there is a high autocorrelation between consecutive data points. This indicates that it is possible to predict the demand for a specific time based on historical data. In the following sections, the optimal memory length, in other words, the number of data points used to predict, will be analyzed for each implemented model. Furthermore, it is shown that there is no strong correlation between the temporal change of the demand of different units. This is an interesting observation because in the first case, it could be reasoned that the demand of the units could impact each other. For example, when a unit's maximum occupancy rate is reached, it could be reasoned that the demand for units with a comparable volume in the vicinity of the fully booked location increases certain time steps later. However, the data analysis showed that this is generally not the case. By analyzing the cross-correlation between the different time series, it is also proven that the temporal change of the units is non-trivial, which means that each time series behaves uniquely over time. After analyzing the distribution of demand for the different units, it can be concluded that the demand depends on the geographic location of the storage building, the unit volume, and the floor on which the units are located. These variables are, therefore, taken into account when making predictions in the rest of the thesis.

5 Predictive Algorithms

In this chapter, it will be discussed which models are implemented and how they are implemented. This includes the definition of the model's input and output parameters, the outlining of each parameter, as well as how each parameter affects the performance. The Markov chain model, LSTM, ConvLSTM are the models addressed in Section 5.1, 5.2, and 5.3, respectively.

5.1. Markov Chain

The Markov chain is a predictive algorithm that utilizes the information of a single link in the network to predict the demand ratio of the link. Therefore, the algorithm should be run for each link in the network separately. The average performance is then calculated by taking the average error of all predictions. The Markov chain consists of states, and the states are connected by transition probabilities, indicating the probability of transitioning from one state to the next state. Markov simplifies the problem by assuming that the next state only depends on the present state. In this project, the states of the Markov chain represent the demand ratios. Therefore, the transition matrix contains the probability of transitioning from one demand ratio to the next. In this section, it will be explained how the transition probabilities are calculated.

The Markov chain contains one variable that needs to be defined beforehand, the memory length. The memory length, defined as l, is the number of demand ratios taken into account when predicting the next demand ratio. When $l = 1$, only one value is used to predict the next value. This indicates that the states are defined by a single demand ratio. When $l > 1$, the states are defined by a sequence of demand values of length l. l is an important variable and will be optimized by testing the performance on the validation set while using different values for l. The l that results in the lowest average error is used to predict. The results of this optimization will be discussed in Section 6.2.1. It is important to note that increasing l results in a larger number of possible states. Therefore, the memory length needs to be appropriately chosen depending on size of the dataset.

It is relatively straightforward how to apply the Markov chain on this dataset. Next, it will be briefly highlighted. As already mentioned, l needs to be properly chosen depending on the size of the data. In this project, the number of data points is equal to 1460, and therefore l should be kept relatively small. Based on the chosen value of l, the time series X with length n can be split into n - l sequences. Thus, the i-th sequence contains $\{X_i, X_{i+1}, .., X_{i+l}\}$ where $\{X_i, X_{i+1}, .., X_{i+l-1}\}$ is used to predict X_{i+l}. Based on these sequences, the transition probabilities in the transition matrix are updated. Then, for each sequence in the test set, the predicted value is computed as the state to which the transition probability from its previous state is the highest. It is worth mentioning that the predicted value can never be a value that does not exist in the time series. In Section 6.1.3 it is discussed how the predictions are evaluated. Section 6.1.1 discusses how the data is split up into a training and test set, where the training set is used to calculate the transition probabilities of the transition matrix, and the test set is used to evaluate the performance of the model. For a detailed explanation of the algorithm, one can read the pseudocode of the Markov chain in Appendix A.1.

5.2. LSTM

To find out if a more complex model outperforms the relative simple stochastic model defined in Section 5.1, the LSTM model is implemented. The LSTM is a version of a Recurrent Neural Network (RNN) that is created to address the problem of RNNs lacking long-term memory. The LSTM consists of a cell state, hidden states, and three types of gates used to regulate the flow of information in the LSTM unit. A schematic overview of the LSTM is shown in Figure 5.1. The names of the three gates are the forget gate F_t, input gate I_t, and output gate O_t. Besides that, it also contains hidden states H_t, cell states C_t and the input X_t. In Equation 5.1, the key equations used by the LSTM are shown, with $\odot$ representing the Hadamard product. The indexed parameter W represents the weight parameters for each state, while the indexed parameter b represents the bias. By using Equation 5.1 and the schematic overview, the working of the LSTM will be briefly elaborated.

$$\begin{aligned}
F_t &= \sigma(W_{xf}X_t + W_{hf}H_{t-1} + b_f)\\
I_t &= \sigma(W_{xi}X_t + W_{hi}H_{t-1} + b_i)\\
\tilde{C}_t &= tanh(W_{xc}X_t + W_{hc}H_{t-1} + b_c)\\
C_t &= F_t \odot C_{t-1} + I_t \odot \tilde{C}_t\\
O_t &= \sigma(W_{xo}X_t + W_{ho}H_{t-1} + b_o)\\
H_t &= O_t \odot tanh(C_t)
\end{aligned} \tag{5.1}$$

F_t is used to determine how much information/data needs to be kept or thrown away in the cell state. In the first line of Equation 5.1, it can be seen that F_t is computed using the input X_t, and the previous hidden state H_{t-1}, both multiplied by the trained weight matrices W_{xf} and W_{hf}, respectively. The result is then summed with the bias term b_f, which is the input to the sigmoid activation function (σ) that computes the forget value. The sigmoid function ensures that the output is always between 0 and 1. In this way, it can be determined how much information is forgotten. A value close to 0 means that more information is forgotten, and a value close to 1 means more information will be kept.

The input gate (I_t) determines what information needs to be stored in the cell state and can only add information. As can be seen from the second line of Equation 5.1, the value of I_t is computed similarly but with different weight vectors. In the third line of the equation, it can be seen how the candidate memory ($\tilde{C}_t$) is computed. $\tilde{C}_t$ is computed using the tanh activation of the summation of the input (X_t) and the previous hidden state (H_{t-1}) both multiplied by their learned weight parameter plus the addition of the bias term. $\tilde{C}_t$ is then used in line 4 of the equation to compute the cell state (C_t).

C_t represents the long-term memory and is initialized with zeros. C_t is updated by taking the Hadamard product between the forget state (F_t) and the previous cell state (C_{t-1}) plus the Hadamard product of the input gate I_t and $\tilde{C}_t$. The fifth line in the equation shows how the output gate (O_t) is updated and is used to determine what information will be stored in the next hidden state. It is computed using the input (X_t) and the previous hidden state (H_{t-1}). An output gate value close to 1 means that all memory information is passed through the cell, while a value close to zero means that no information is passed through the cell.

The final line of the equation shows how the hidden state is updated by computing in the Hadamard product between the output gate O_t and the tanh activation function of the state (C_t). Figure 5.1 gives a schematic overview of the process. Next, it will be explained how the LSTM is used to predict future occupancy rates.

The Keras [43] library is used for the implementation of the LSTM. First, the model's input must be defined. The training data must be translated to fit the input format of the LSTM, which is a 3D tensor (batch, timesteps, feature). Because no information other than the time series data is used for the predictions, the number of features equals 1. The number of timesteps is equal to the number of data points used for the predictions and is equal to the memory length defined as l. Different values for l will be used to see how the memory length influences the model's performance. The number of batches depends on the memory length and the size of the training data. The ratio between the size of the training and test set will be discussed in more detail in Section 6.1.1.

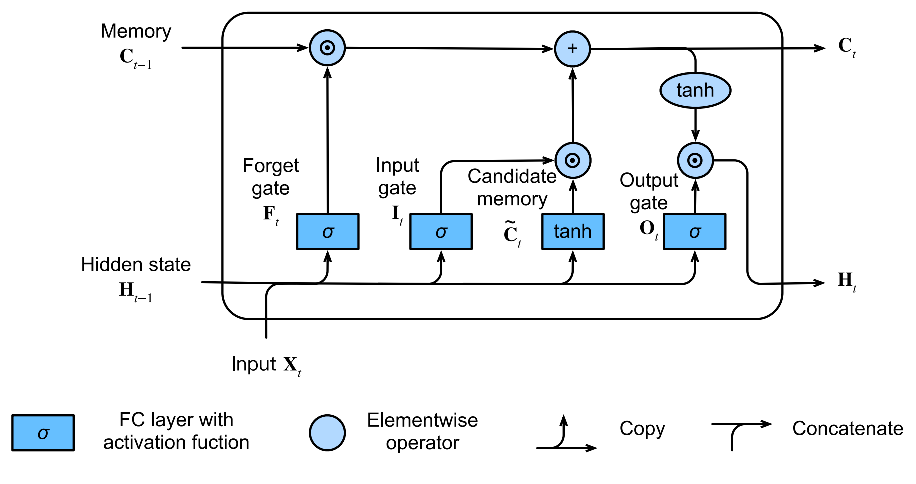


Figure 5.1: Schematic overview of a LSTM cell [1].

Many parameters have an impact on the performance of the model. In Section 6.1.2, it will be explained how the optimal set of hyperparameters is found systematically. The most important hyperparameters of the LSTM are defined below:

- Memory length - number of historical data points used to predict
- Number of layers - number of stacked LSTMs
- Number of units - dimensionality of the output space
- Batch size - number of training samples used in one iteration
- Epochs - the number of times the entire training set is worked through the algorithm
- Learning rate - the amount of change the weight parameters of the model are updated
- Dropout rate - fraction between 0 and 1 which represents the fraction of units to exclude during training, and this can be used to avoid an overfitting model
- Optimizer - the optimization function used by the model to find the optimal weight parameters
- Activation function - the activation function used in the layers to define what information should be used in the next layer or cell

Note that it is important to select the values used for these parameters carefully. In Section 6.2.2, an in-depth study of the hyperparameter optimization, as well as the performance of the LSTM, will be presented.

5.3. ConvLSTM

The previous section explains how the LSTM can predict the occupancy rate of future time steps, which uses only information of a single link in the network to predict. The convolution LSTM (ConvLSTM) is an extended version of the LSTM. Besides the temporal information captured by the LSTM, the ConvLSTM also tries to capture the spatio-temporal dependencies in the network. In other words, the correlation among the links is taken into account when predicting. This approach is the first method that uses the information of multiple graph snapshots rather than the information from a single link. By comparing the performances of LSTM and ConvLSTM, it can be concluded whether using this extra dimension improves the model's performance.

The main functionality of the ConvLSTM is similar to that of the LSTM. The differences, on the other hand, will be highlighted next. The input of the ConvLSTM is a 5D tensor of the following format (samples, time, channels, rows, cols). The height and width of the adjacency matrix represent a graph snapshot, which is equal to the values of the rows and cols. Often, the ConvLSTM is used in image processing. Then, the number of channels equals 3 representing the red, green, blue (RGB) values for each pixel in the image. However, in this project, the number of channels is equivalent to 1 since the demand consists of only one value at a given point in time. The time variable is equal to the memory length l and needs to be fine-tuned. The number of samples depends on l and the size of the training set. The key equations of the ConvLSTM are almost identical to the equations of the LSTM. They are shown in Equation 5.2 with * being the convolution operator and $\odot$ representing the Hadamard product.

$$\begin{aligned}
F_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot C_{t-1} + b_f) \\
I_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot C_{t-1} + b_i) \\
\tilde{C}_t &= tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\
C_t &= F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \\
O_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \odot C_{t-1} + b_o) \\
H_t &= O_t \odot tanh(C_t)
\end{aligned} \tag{5.2}$$

The main difference between Equation 5.1 and Equation 5.2 is that the matrix multiplication is replaced by the two-dimensional convolution when computing the values of the gates. Because the convolution operator is used in the horizontal and vertical direction, the model can capture both spatial and temporal features. Besides the replacement of the matrix multiplication, it can be noticed that an additional term is used when computing the values of the input, forget, and output gates. In more detail, the Hadamard product between W_{cf} and C_{t-1} is used when computing F_t, the Hadamard product between W_{ci} and C_{t-1} when computing I_t, and the Hadamard product between W_{co} and C_{t-1} when computing O_t. These additional terms are also used by a variant of the LSTM; the Fully Connect LSTM (FC-LSTM). The additional connections also called peephole connections, allow the gate layers to access the cell state. Therefore, the gates can compute their values based on the incoming inputs and the previous state. A schematic overview of the ConvLSTM is shown in Figure 5.2.

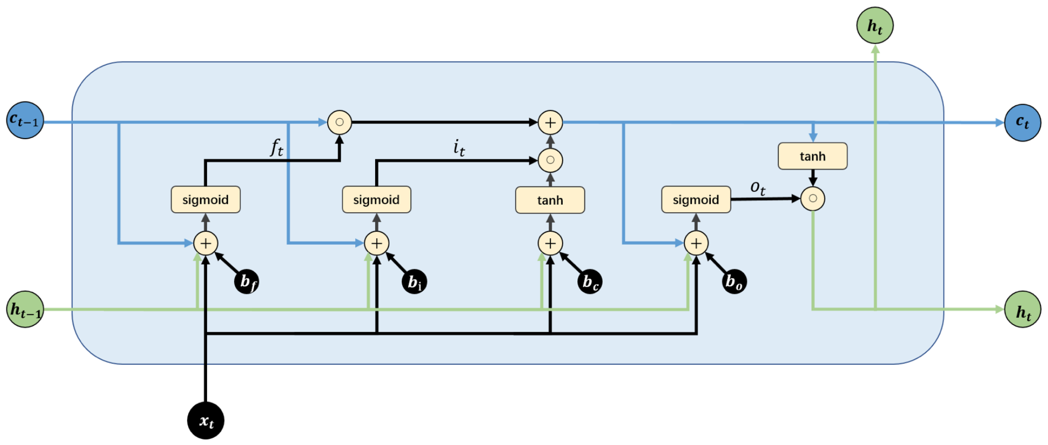


Figure 5.2: Schematic overview of a ConvLSTM cell [2].

Again, the Keras library is used to implement the ConvLSTM model. There are several parameters used by the model that need to be defined beforehand. Apart from the parameters already defined in Section 5.2, the most important parameters are:

- Filters - the number of output filters in the convolution
- Kernel size - represents the dimension of the convolution window
- Stride - tuple containing the height and width the filter is shifted
- Padding - the name of the padding method needs to be specified (if any padding method is required)

Note that the padding variable needs to be equal to "same" because the input dimension needs to be equal to the output dimension. Otherwise, the predictions can not be linked to a location or unit type. The impact of these parameters will be analyzed when optimizing the hyperparameters, and the results of this optimization will be discussed in Section 6.2.3.2.

6 Experiments

To compare the performances of the different models, it is necessary to analyze their performances systematically. The evaluation procedure will be explained in this chapter. First, the configuration of the models is explained, and this describes how the data is split into training and test sets. Furthermore, it explains the optimization of the hyperparameters and the metrics used to evaluate the performance. After that, for each model, it will be elaborated on how it performs.

6.1. Experimental Setup

The configuration for the implemented models will be discussed in this section. Three design choices could have a significant impact on the performance of the model. First, the training and test set must be carefully chosen. Besides that, the hyperparameters used by the models impact the models' outcomes and should therefore be optimized systematically. And finally, the metrics used to measure the performance of the models should be defined properly. This section will describe the entire process and explains why certain decisions were taken.

6.1.1. Selection Train and Test Set

As mentioned already several times, the data must be divided into a training and a test set. The training set is used to train the model, and this includes finding the optimal hyperparameters. The test set is used to evaluate the final performance of the model. To evaluate the performance of the model objectively, the test set must be excluded from the training set. Furthermore, each model should be trained with the same training data and evaluated with the same test data.

Overfitting and underfitting are two well-known and important definitions in data science. Overfitting is a term used to describe a model that fits the training data too well and does therefore not perform well on the test set. Underfitting is a term used for having a model that cannot perform well on both the training and test set. Without cross-validation, the likelihood of creating a model that over or under fits increases because it is only trained and tested once. Therefore, cross-validation should always be applied to increase the reliability of the experiment.

Aside from the training and test set, cross-validation introduces another set, the validation set, which is a subset of the training set. This subset is excluded from training and used to evaluate the performance of the trained model. After analyzing the results on the validation set(s), it can be determined what the optimal hyperparameters are. One of the most commonly used cross-validation methods is k-fold cross-validation [44], with k being a parameter that must be defined beforehand. K-fold cross-validation divides the training data into k parts (k-folds), and each fold is used for validation once. Because the model is trained and evaluated on different parts of the dataset, it can be seen whether the performance on the different folds differs significantly. Based on these results, the reliability of the training phase and the set of hyperparameters used can be determined. Figure 6.1 shows a schematic overview of k-fold cross-validation, with $k = 5$ and the orange folds representing the validation folds. Finally, the independent test set (represented by the yellow block in Figure 6.1) is used to evaluate the model's final performance using the optimal hyperparameters found using cross-validation.

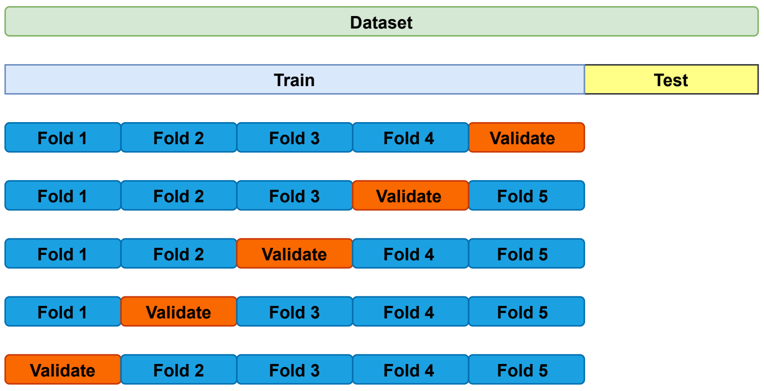


Figure 6.1: Schematic overview of k-fold cross-validation.

The data in this project is time-dependent. Therefore, k-fold cross-validation is not a practical method that can be applied because it does not make sense to use future data to predict the past. For example, when fold 3 of Figure 6.1 is used as a validation set, the model is trained with folds 1, 2, 4, and 5, respectively. This implies that future knowledge (folds 4 and 5) is used to forecast the past (fold 3). To overcome this problem, a slightly different but comparable approach is used.

The technique that solves the above-defined problem uses a sliding window to create several folds to train the model, find the optimal hyperparameters, and evaluate the performance of the final model. However, when creating the folds, the data's temporal dependency is preserved. This approach uses several variables that need to be defined beforehand. First, the ratio between the training and test set needs to be defined. A commonly used ratio is 70% for training and 30% for testing. This ratio is also used in this project. Thus, each model is trained with the first 70% of the data, while the last 30% is used to evaluate the performance of the models. Then, the size of the sliding window, W, needs to be chosen. The sliding window can be defined as the number of data points used in each training fold. Finally, the step size P needs to be defined, and P represents the number of data points W is shifted each iteration. The total number of folds depends on the values selected for W and P.

A brief example is shown to demonstrate the working of this technique. In this project, the length of the time series X containing weekly data is roughly equal to 215. The first 70% of the data is used for training while the last 30% is used for testing. Thus, around 150 data points are used for training represented by T, while the other 65 data points are used for testing. Given that P = 1 and W = 105 (70% of the training set) the total number of folds is equal to F = $\frac{T-W}{P} = \frac{150-105}{1}$ = 45 folds. The first fold, $\{X_0, X_1, ..X_{104}\}$, is used as a training set while the next data point X_{105} is used as a validation set. Then the sliding window is shifted by P. The second fold, $\{X_1, X_2, ..X_{105}\}$ is used as a training set, and X_{106} is used as a validation set. In general, $\{X_i, X_{i+1}, ..X_{i+W-1}\}$ is the training set of fold i and X_{i+W} is the validation set with $0 \leq i \leq F$. A schematic overview of this approach is shown in Figure 6.2. It is worth mentioning that when $P = 1$, the sliding window is shifted one single data point, resulting in many folds. Because training more complex models generally result in long runtimes, it is necessary to determine whether $P = 1$ is feasible. Otherwise, P should be increased to reduce the number of folds.

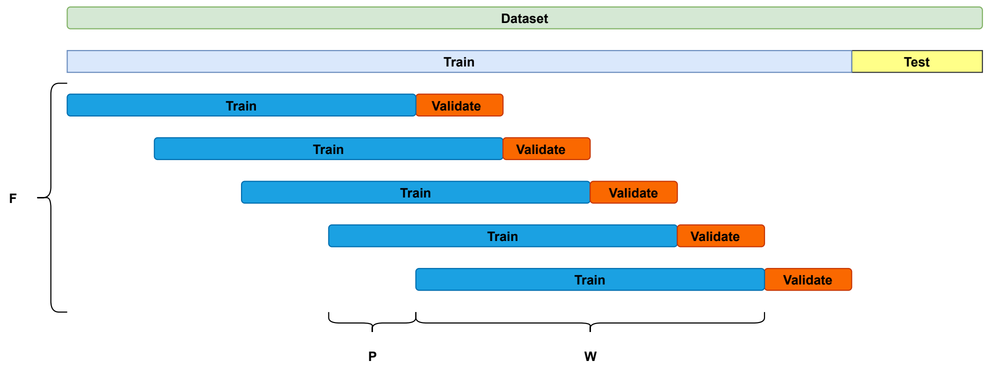


Figure 6.2: Schematic overview of sliding window cross-validation.

6.1.2. Optimization of Parameters

All implemented models base their predictions on tunable parameters (e.g., memory length, activation function). The chosen variables impact the predictions made. For this reason, it is important to set the parameters carefully in order to obtain high accuracy. As shown in Section 5, the more complex models have many hyperparameters to optimize. Therefore, the hyperparameters need to be optimized systematically.

The grid search technique is frequently used to find these optimal parameters. To apply this technique, one should first construct a set of values for each hyperparameter the model uses. Then, for each combination of parameters, a model is created using these parameters. This model is trained on the training set and evaluated on the validation set, and this process is repeated for each fold. The grid search identifies a set of parameters for which the model performs optimally on average on all folds, and the final performance of the models is computed using this optimal set of parameters.

Because training models is a time-consuming process in general, it must be determined whether it is possible to optimize the models for each link in the network. When this is not feasible, a randomly selected set of time series is used to find the parameters that, on average, result in the best performance. This is only relevant for the approaches that predict the information of a single link.

6.1.3. Evaluation

This section describes the evaluation procedure used in this project. This includes a way to quantify the performance of the different models and a method for computing the upper bound of the predictability of temporal networks.

6.1.3.1 Evaluation Metrics

In previous sections, it is discussed how the models are trained and evaluated. However, no metric has yet been defined to quantify the performance of the different models. A frequently used metric to measure the performance of a model that predicts continuous numerical values is the Mean Absolute Error (MAE). The formula to compute the MAE is given in Equation 6.1, with n being the total number of predictions made, y_i the actual value and $\hat{y}_i$ the predicted value.

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n} \tag{6.1}$$

MAE calculates the average error by computing the absolute differences between the predicted and the actual value. Therefore, a smaller MAE is better. Because the absolute difference between the expected and actual values is computed, the sign of the error is irrelevant. This also holds when predicting the demand. It is a good metric to use when the data contains outliers because each error contributes equally to the total error. The error quantity is equal to the quantity of the input data, which makes it easy to understand what the error means. Because of these properties, the MAE is an appropriate metric.

Another metric often used is the Mean Squared Error (MSE) which can be found in Equation 6.2, where n represents the total number of predictions made.

$$MSE = \frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n} \tag{6.2}$$

MSE calculates the squared difference between the predicted values and the actual observed values. Because the difference between the predicted value and the actual value is squared, larger errors are penalized more. This is the main difference with respect to the MAE. When the data consists of many outliers, the MSE could incorrectly represent the model's performance because the outlier errors heavily impact the evaluation. For this reason, it is critical to analyze whether the dataset consists of many outliers before selecting the evaluation metric. The quantity of the input data is different from the quantity of the MSE. To overcome this issue, the RMSE is frequently used. The RSME is the squared root of the MSE, as can be seen in Equation 6.3.

$$RMSE = \sqrt{MSE} \tag{6.3}$$

It is also possible to compute the accuracy of the models. In the following section, it is explained why it is necessary to compute the accuracy of the models. Because the complex models, predict floating values with a lot of decimal places, the predictions made by the model should be first discretized to the closest element observed in the network. Then, the accuracy can be computed by Equation 6.4, where I represents the Boolean indicator function which is equal to 1, if in this case, $\hat{y}_i = y_i$, is true, and 0 otherwise. And $\hat{y}_i$ represents the discretized prediction.

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i = y_i) \tag{6.4}$$

In this project, the RMSE is mainly used to evaluate the model's performance. However, as will be shown in Section 6.3, the final performances will be evaluated using the MAE, RMSE, and accuracy. Evaluating the performance of the models using multiple evaluating metrics increases the reliability of the experiment. Besides that, it can gather interesting insights about the prediction behavior of the models.

6.1.3.2 Theoretical Upper Bound

In [2], an entropy-rate-based framework is proposed to compute the upper bound of the predictability of real temporal networks. This value is the upper bound of the prediction accuracy of any prediction algorithm. We will compare the accuracy of the three predictive algorithms with the theoretical upper bound to see whether there is still space to further improve these prediction methods.

To compute the theoretical predictability, an $m \times n$ matrix M needs to be created, where m is the total number of possible links in the network and n is the total number of time steps. Thus, each column represents one graph snapshot. Then, the matrix is filtered and ordered. As proven in [2], this does have a negligible impact on the predictability of a network. It is crucial to filter the matrix because many real temporal networks are very sparse, which means that the matrix M contains many rows which only contain many zeros. This leads to high predictability, and for this reason, the matrix is filtered such that it only contains links that are present in more than 10% of the graph snapshots.

Because the matrix M can be seen as a sequence of random vectors, the overall entropy rate can be computed to quantify the level of randomness contained in the matrix. The entropy rate is calculated using a variety of mathematical equations. The main idea behind this computation is to calculate the recurrence of different patterns within the matrix M. Based on the entropy rate, the upper bound of the predictability of the network can be computed. A detailed explanation, as well as the equations to compute the upper bound, are given in [2].

6.2. Results

In this section, the results of the different models will be presented and analyzed. Also, the challenges faced during the analysis will be discussed. The performance of the Markov chain model is first discussed in Section 6.2.1, followed by an analysis of the performance of the LSTM and the ConvLSTM in Section 6.2.2 and 6.2.3. Finally, the performances will be compared to each other and to the upper bound of the predictability of the network.

6.2.1. Markov Chain

The challenges encountered during the performance analysis of the Markov chain will be briefly discussed in this section, followed by the solutions to these challenges. The results are presented after that.

6.2.1.1 Challenges

The Markov chain model is used as a baseline method in this project. As already mentioned in Section 5.1, an important variable that needs to be set beforehand is the memory length defined as l. It is relatively straightforward that the number of states increases when the memory length is increased. Therefore, the memory length needs to be properly chosen depending on the size of the dataset. Because a too large number of states will result in an overly sparse transition matrix. For illustration purposes, the number of states is plotted against the memory length in Figure C.1a in Appendix C.1.

Discretization is a commonly used method to reduce the number of states. The continuous data is discretized by dividing it into intervals, also called bins. The number of bins needs to be defined beforehand. The first method, uniform discretization, ensures that each bin has an equal bin width. In contrast, the second method, quantile discretization, ensures that the number of data points within each bin is roughly equal. The influence of the discretization approaches mentioned above is compared to each other to see how they affect the structure of the Markov chain and the performance.

In Figure 6.3, the number of states is plotted against the memory length for each discretization method. It is worth noting that the line with the label "Non-discretized" represents the Markov chain when no discretization method is used. However, as also mentioned in Section 5.1, the demand values representing the states are rounded to two decimals before creating the Markov chain. Rounding the data to two decimals can also be seen as a way to discretize the data. In this project, two different values are used for the number of bins, 5 and 10, respectively. Choosing larger values will again result in too many states, especially when l increases. Because the number of bins is kept small, it is important to ensure that the problem is not oversimplified. Later in this section, it will be shown how this simplification affects the performance. The plot shows that using uniform discretization with 10 intervals results in a comparable number of states as the non-discretized Markov chain. When using only 5 bins, the number of states for $l = 8$ is above 100 for all discretization methods. Having that many states will still result in a very sparse transition matrix. For this reason, it is decided to limited l to 4. When $l = 4$, the Markov chain consists of around 60 states which seems like a decent average number of states. Next, the performance of the different configurations of the Markov chain will be presented.

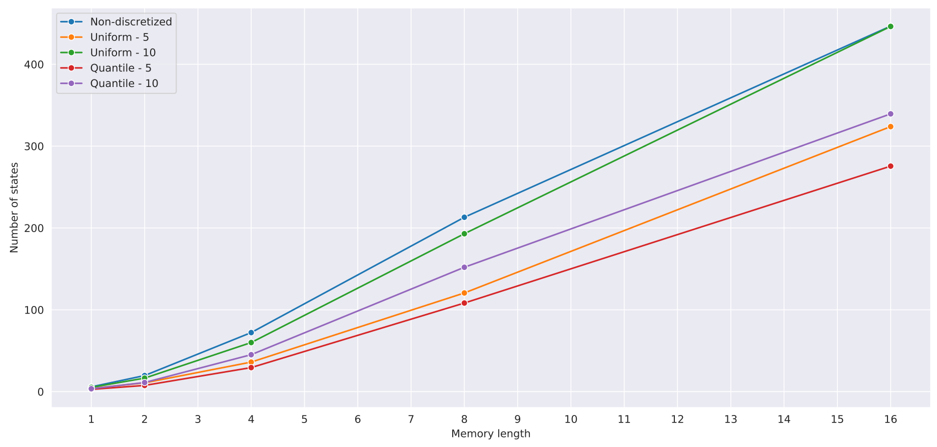


Figure 6.3: The number of states of the Markov chain after discretization plotted against the memory length.

6.2.1.2 Performance

The training folds are used to determine the optimal discretization method and memory length by comparing the different configurations' average performance. In Figure 6.4, the RMSE averaged over all units is plotted against the memory length. It is worth noting that the discretized data is compared to the actual values, instead of the discretized data, when calculating the error rate of the models. It is also possible to transform the discretized prediction back to the closest original value in the time series. The difference between these two performances will be presented in Section 6.3. As can be seen, the uniform discretization methods outperform the quantile discretization approaches by more than a factor of two. The uniform discretization approach with 10 bins has the lowest RMSE value for all memory lengths. Hence it can be stated that this discretization method is most suitable. The large difference in performance between the various discretization methods is most likely due to the bin widths determined when discretizing the data. Based on the performance of the different discretizing techniques, it can be concluded that the uniform discretization better fits the data structure due to the lower error. This technique will be used to discretize the data.

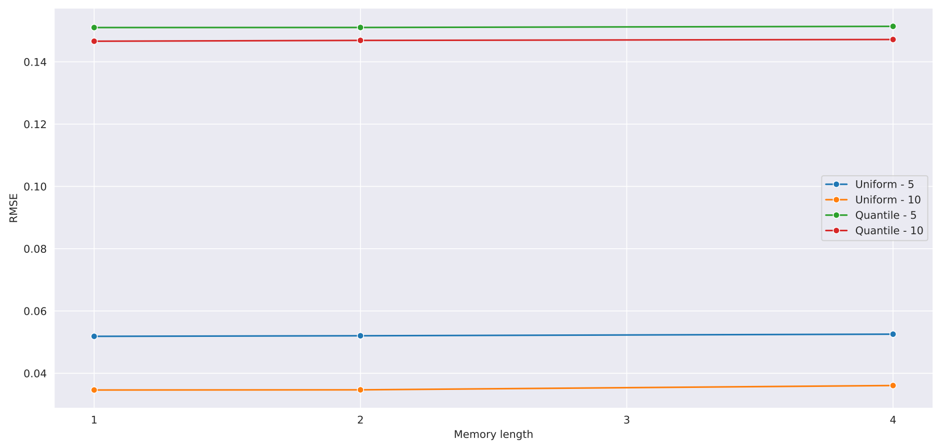


Figure 6.4: Performance of the Markov chain model plotted against the memory length.

It is interesting to see that the memory length does have a negligible impact on the performance of the Markov chain model in terms of RMSE as can be seen in Figure 6.4. Next, it will be analyzed whether the memory length impacts the performance on the unseen test set.

In Figure 6.5, both the training and test error is plotted against the memory length using the optimal discretization method. The blue line represents the average performance on the training set, while the orange one represents the test set's average performance. Due to the minimal range in the y-axis, the performance on the test set is comparable to the performance on the training set. This highlights that the training data is a good representative of the entire dataset, meaning that the data samples contained in the training set are comparable to the samples in the test set. The RMSE is minimized when $l = 1$, resulting in an RMSE of 0.035, indicating that using a single data point to predict the next occupancy rate is the optimal configuration for the Markov chain model. Because the baseline model performs relatively well, it will be investigated why it performs so well and why the differences in performance are negligible for the different memory lengths. The results of this analysis will be presented in Section 6.2.1.3.

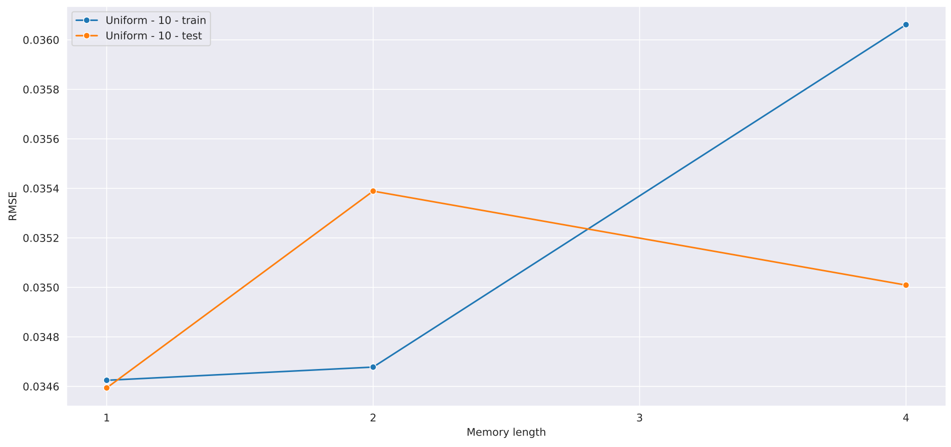


Figure 6.5: Performance of the Markov chain model on the training set compared to the performance on the test set.

6.2.1.3 Technical Analysis on Daily Data

The transition matrix of the best performing discretization method (uniform discretization with 10 bins) is studied to understand the Markov chain model's predictions better. First, the number of reachable states will be analyzed. Analyzing the number of reachable states is an interesting characteristic to analyze because this could indicate the generalization of the Markov chain. When a single state can be reached on average, the conclusion could be that the number of states is too small or too large. In Figure 6.6, the number of reachable states is plotted against the memory length. From this figure, it can be seen that for a smaller l, the number of reachable states is higher. When $l = 1$, around 3 states can be reached. This number decreases extensively when l increases. When $l = 4$, the number of states that can be reached is close to 1. This is because when the memory length increases, the number of unique sequences of length l, in other words, the total number of states, grows as well as already shown in Figure 6.3.

Secondly, it is interesting to know what the average maximum transition probability per state is. Therefore, the highest transition probability is stored for each state. The highest transition probability represents the change of jumping from one state to another state. The higher this number is, the more this transition occurred in the training set, and the more evidence is provided that this transition will occur in the future. However, when the maximum probability is equal to 1, it could also be the case that the transition between two states occurred only once during training. To see how the maximum transition probability is related to the memory length, the average maximum probability is computed and plotted against the memory length in Figure 6.7. The average maximum probability is larger for smaller l. When $l = 1$, the average maximum transition probability equals 0.86 which is extremely high. Furthermore, for all l, the average maximum transition probability is above 0.83. Having an average maximum probability of at least 0.83 and knowing that the average Markov chain contains less than three transition probabilities greater than 0 indicates that the Markov chain contains one transition probability with a very high probability and the other transition probabilities are relatively small.

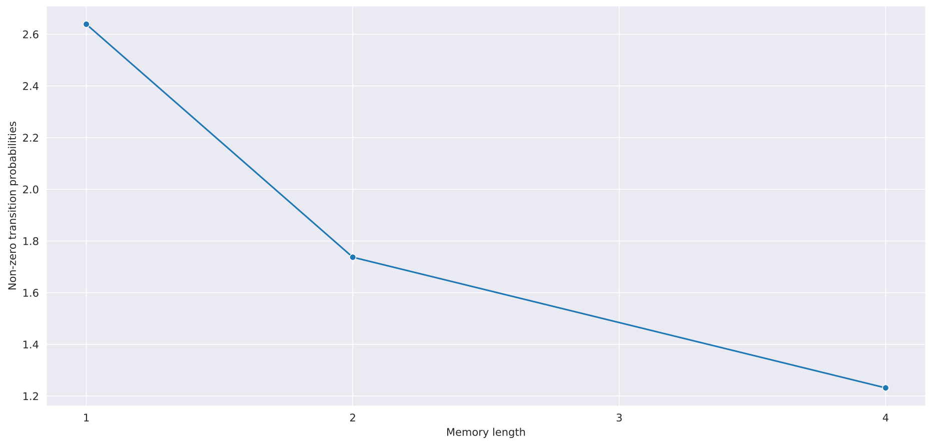


Figure 6.6: The average number of states reachable plotted against the memory length.

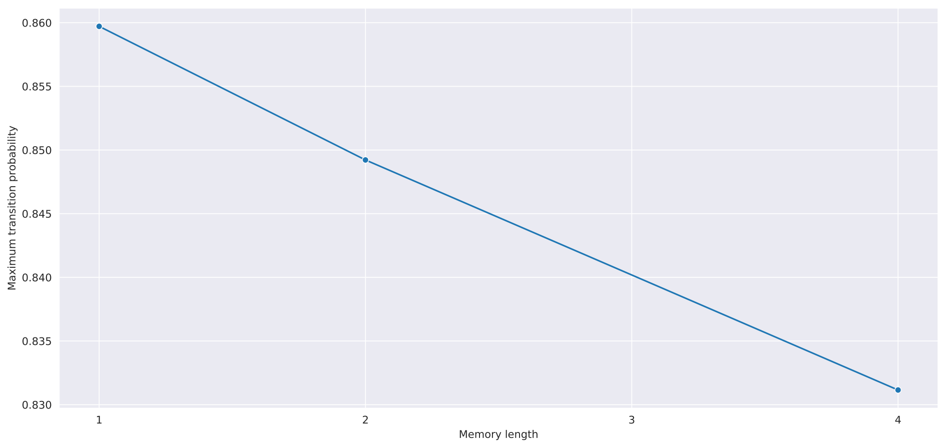


Figure 6.7: The average maximum transition probability of the states plotted against the memory length.

To see how the predicted value differs from the input, the difference between them is analyzed. After creating several plots to compare the input and output, an interesting insight was gathered. The prediction was frequently found to be equal to the input. In other words, the self-transition probability of a state often contained the highest transition probability. Consequently, the number of times a state's self-transition had the highest transition probability was computed. In Figure 6.8, the fraction of states containing the highest transition probability to itself is plotted against the memory length. As can be observed, the fraction is larger than 0.9 for $l = 1$. This means that more than 90% of the states predict their own value for the next time step when using a single data point to predict. Figure 6.7 already showed that the average highest probability per state was around 0.86 when $l = 1$. This provides statistical evidence that confirms that the demand does not frequently change daily. For this reason, it has been decided to aggregate the daily data to weekly data by taking the average of seven occupancy rates. Because this new dataset probably changes more frequently, it is more challenging to predict the occupancy rate of the next time step. Furthermore, having a model that predicts the same occupancy rate equal to the previous time step provides limited benefits for the company. Therefore, the aggregated dataset based on weekly input data will be used for the remaining models.

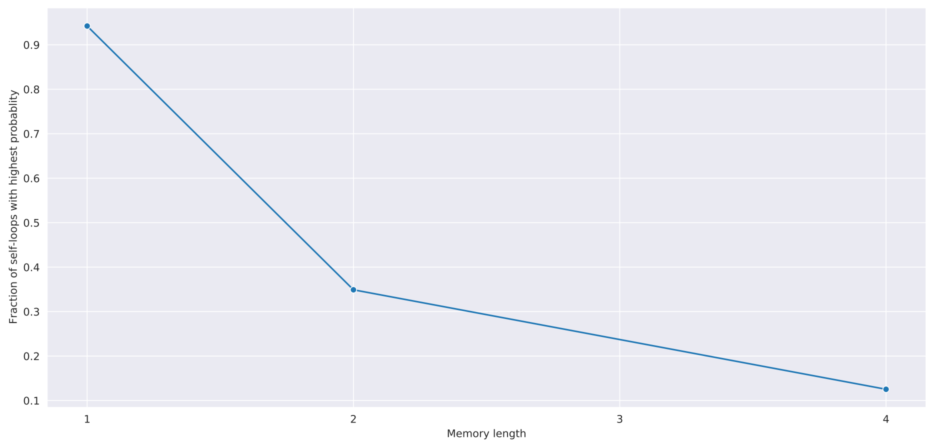


Figure 6.8: The fraction of states that contain the highest transition probability to itself plotted against the memory length based on daily data.

6.2.1.4 Technical Analysis on Weekly Data

To see whether the aggregated dataset changes more frequently, a similar plot as Figure 6.8 is created. Figure 6.9 contains the result of this. In this figure, the fraction of states where the self-transition probability has the highest probability is plotted against the memory length. This figure proves that the aggregated dataset changes more frequently because when $l = 1$, we see that around 45% of the states predict their own value for the next time step. This fraction was around 90% when using the daily dataset. This observation highlights that the demand for the new dataset changes more frequently, making it more interesting and challenging to predict.

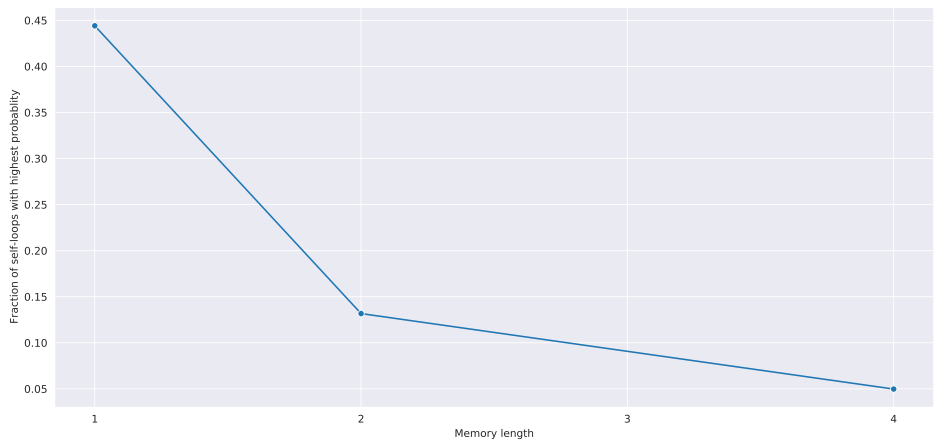


Figure 6.9: The fraction of states that contain the highest transition probability to itself plotted against the memory length based on weekly data.

Because the data is aggregated to weekly data, the size of the dataset decreases. A time series consists of approximately 210 data points when the data is aggregated. Figure C.1a showed that the number of states grows approximately linear to l. Now that the size of the dataset is decreased, it needs to be analyzed whether it is possible to use the same memory lengths. Again, the number of states is plotted against the memory length. Besides that, the average number of reachable states is plotted against the memory length. Based on the results of these plots, it is shown that using a large memory length results in a large number of states and thus an overly sparse transition matrix. Therefore, the memory length is limited to 2. One can read more about this analysis in Appendix C.1.

Table 6.1 contains the final performance of the Markov chain model on the weekly dataset. The table shows the average performance in terms of the RMSE and the standard deviation when using a memory length of 1 and 2. Both the performance on the training set and test set are shown. It can be seen that the average performance on the training set is comparable to the performance on the test set. This does not hold for the standard deviation. We see that the standard deviation on the training set is smaller than the standard deviation on the test set for both memory lengths. This indicates that for some time series, the test set contains more unpredictable events which are not contained in the training set. When comparing the performance of the different memory lengths, we can conclude that the memory length barely impacts the performance. $l = 2$ slightly outperforms $l = 1$ because of the lower training error and test error as well as the lower standard deviation. However, the difference between them is not significant. This only holds because we limited the memory length to 2. When a larger memory length is used, it will probably result in poor performance because of the sparsity of the transition matrix. Using larger memory length also increases the complexity of the models, while the performance barely increases. This is another reason why the memory length is kept small.

In Figure 6.5, we saw that the optimal model was able to get an RMSE of 0.035 on the test set. For the weekly dataset this value is equal to 0.053 as can be seen in Table 6.1. For this dataset, this difference can be considered significant. This verifies the assumption mentioned above that when the demand changes more frequently, it is harder to predict for the Markov chain model. In Section 6.3, the performance of the Markov chain will be compared to the other implemented methods. By comparing the performances, we can research whether more complex models can predict more accurately, and thus, resulting in a lower RMSE. In Section 6.3.2, the actual predictions will be visualized and compared to the actual values. In this way, we can observe what the prediction behavior of the model is.

Table 6.1: The final performance of the Markov chain.

	Training error		Test error	
Memory length	Mean	Std	Mean	Std
1	0.046	0.033	0.056	0.057
2	0.045	0.029	0.053	0.048

6.2.2. LSTM

The difficulties encountered during the implementation of the LSTM will be discussed in this section. Also, the results of the hyperparameter optimization are presented.

6.2.2.1 Challenges

As shown in Section 5.2, the LSTM has many parameters that need to be fine-tuned. Ideally, the parameters should be optimized separately for each time series (756 in total) because the time series change uniquely over time. The total number of parameters that need to be fine-tuned is equal to 8. If three different values are tested for each parameter, the number of parameter combinations used by the grid-search explodes. Testing three different values for each parameter results in 3^8 = 6,561 possible combinations. Because cross-validation is used, the model is trained on different folds. The number of folds for training is approximately 45. Therefore, if the experiment is run for each time series separately, the number of experiments that should be run equals 756 * 6561 * 45 = 223,205,220 experiments. Running so many experiments is infeasible. As a result, it is necessary to consider ways to reduce the number of experiments.

There are several ways to decrease the number of experiments. First of all, the step size used in the cross-validation procedure can be decreased. That is, the number of data points the sliding window is shifted when creating the folds. In this way, the number of folds is decreased, and therefore the model needs to be trained on fewer folds. Another option would be only to use a subset of all time series to find the optimal parameters and use these parameters when predicting. Lastly, the number of tunable parameters could be decreased. The first way to achieve this is to use a fixed value for several parameters and, thus, excluded them from the optimization. As a result, the number of possible combinations for the parameters in grid-search decreases. To choose an appropriate fixed value for the excluded parameters, it is possible to analyze the impact of the excluded hyperparameters separately. An experiment can be run where the model's default values are used while optimizing the excluded parameters to get insights into how these parameters affect the performance. The last way to reduce the number of experiments is to decrease the number of values tested for each parameter. As mentioned above, an example is given where 3 different values are used for each parameter. When this number is decreased, the number of experiments drops significantly. It is worth noting that by using these approaches a sub-optimal set of hyperparameters is found instead of the optimal set of parameters.

We have decided to apply all of the above approaches to decrease the number of experiments when searching for the optimal set of parameters. The number of folds used for training is roughly equal to 45 when a step size of 1 is used. Because training each model 45 times is very computationally expensive, the step size is set to 3. This results in training the model on 15 different folds. Besides, a subset of the time series is used for the hyperparameter optimization. In total, 10 time series are used to find the optimal hyperparameters. Ideally, this number should be larger. However, it is not possible to increase the number of time series used for training due to time constraints. Lastly, the parameters *epochs* and *batch size* are excluded from the grid-search optimization. But, these two variables are both optimized with the default values of the LSTM to get a global idea of how these parameters affect the performance. The default values of the LSTM can be found in the Keras documentation of the LSTM [1]. Finally, for each parameter (*layers, units, learning rate, dropout rate, activation function, optimizer*), 2 or 3 different values are tested. The result of this experiment will be shown in Section 6.2.2.2.

[1]https://keras.io/api/layers/recurrent_layers/lstm/

6.2.2.2 Hyperparameter Optimization

In this section, one can read how the hyperparameters of the LSTM are optimized. The grid-search technique, which is a brute-force approach, is used to test the impact of the values used for the parameters on the model's performance. For the interested reader, the results of this optimization are discussed in this section.

As previously discussed, the impact of the number of epochs and the batch size will be analyzed before the actual grid-search optimization is run. For the number of epochs the following values are used {5, 10, 20, 30, 40, 50, 60} and for the batch size {2, 4, 8, 16, 32, 64}. Using these values, it can be determined what an appropriate default value is for the batch size and number of epochs. This experiment is repeated for the following memory lengths {1, 2, 4, 8, 16}, defined as l. It is worth noting that only the validation set is used to evaluate the performance of the trained model. The performance of the final model with the obtained optimal hyperparameters will be presented and discussed in Section 6.3.

In Figure 6.10, a heatmap shows the result of the experiment, where each cell depicts the average performance of the different memory lengths tested on a randomly selected subset of time series for the specified number of epochs and batch size. Because the RMSE is used to evaluate the model's performance, a darker cell value indicates better performance. From the figure, it can be seen that a smaller batch size and a large number of epochs perform optimally on average. Besides that, the result verifies that the model's performance highly depends on the selected batch size and number of epochs because of the significant difference in performance. This highlights why it is crucial to analyze the impact of these variables. The best performing configuration uses a batch size of 2 and 60 epochs, resulting in an average RMSE of 0.052. However, when using 10 epochs less, the RMSE is nearly equal. Because the RMSE only decreases with 0.001, we need to determine whether using 60 epochs is necessary.

The poor performance when using a large batch size can be explained because it is harder for the model to learn when it uses more samples per epoch. Therefore, it can be argued that limiting the number of epochs to 60 is why the model does not perform well when using large batch sizes. To verify this assumption, the experiment is repeated using a larger number of epochs, respectively 100, 200, 300, and 400. The result of this experiment is shown in Figure 6.11. The figure shows that increasing the number of epochs does not increase the model's performance when using a small batch size. On the contrary, when using a batch size of 16, 32, or 64, the RMSE of the model decreased significantly. Thus, the conclusion that a large batch size needs more epochs to learn holds. However, the best performing configuration (batch size 2 and 60 epochs) still outperforms the performance when using a large number of epochs by a factor of nearly two. For this reason, the results of the experiments presented next are run with a batch size of 2.

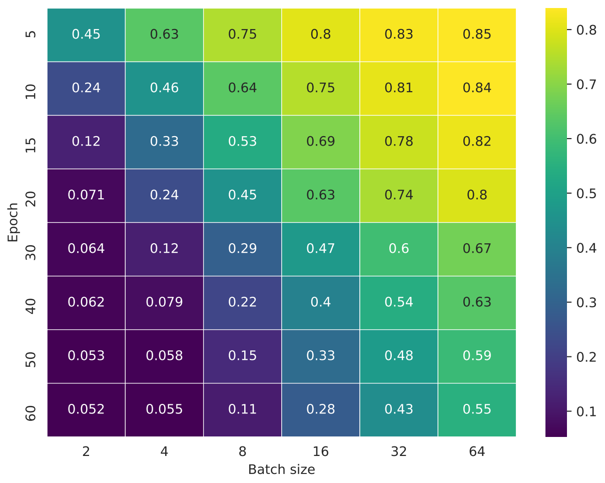


Figure 6.10: Heatmap showing the relation between the performance of the LSTM and the hyperparameters epochs and batch size. The legend (shown on the right-hand side of the heatmap) represents the RMSE. Thus a darker cell value indicates better performance.

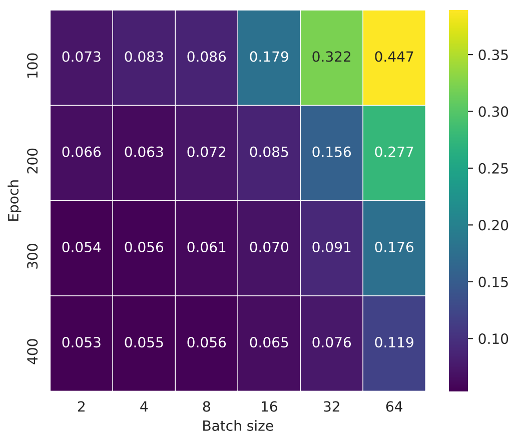


Figure 6.11: Heatmap showing the relation between the performance of the LSTM and the hyperparameters epochs and batch size when using a large number of epochs. The legend represents the RMSE. Thus a darker value indicates better performance.

Because the previously shown heatmaps contain the average performance of the different memory lengths, no conclusions about the best l can be drawn. Therefore, a different plot is created to see what the impact of the memory length is when the optimal batch size (2) is used. This plot is also used to determine whether using 60 epochs is necessary. In Figure 6.12, the performance of various l is plotted against the number of epochs when using a batch size of 2. The above-defined conclusion that a larger number of epochs outperforms a smaller number of epochs holds for all l's because of the only decreasing RMSE when limiting the number of epochs to 60. Besides that, it is clear that when using a larger l, the model can learn faster because the RMSE drops significantly faster than when using smaller ls. This can be explained by the fact that the model has access to more historical data when using larger memory lengths, which enables us to learn faster how the demand changes over time. However, for a larger number of epochs, the performance of the different models is comparable. Thus, although the model can learn faster while using larger l, it does not outperform smaller l for a larger number of epochs. From this figure, it is hard to see which l performs optimally. Therefore the last part of the figure is analyzed to see which l performs optimally.

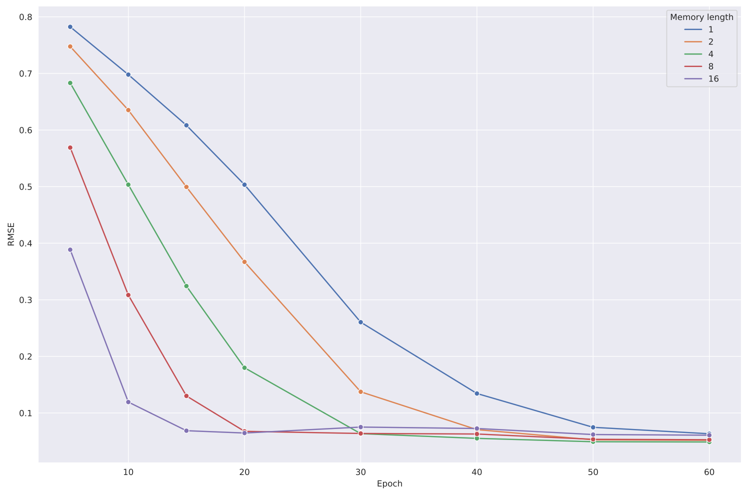


Figure 6.12: The performance of several memory lengths plotted against the number of epochs when using a batch size of 2.

When we zoom in on the last part of the graph (see Figure 6.13), it can be observed that $l = 4$ outperforms the other l. Also, after epoch 50, the performance barely increases for almost all memory lengths. Based on these observations, the grid-search optimization is run with a batch size of 2, 50 epochs, and a memory length of 4.

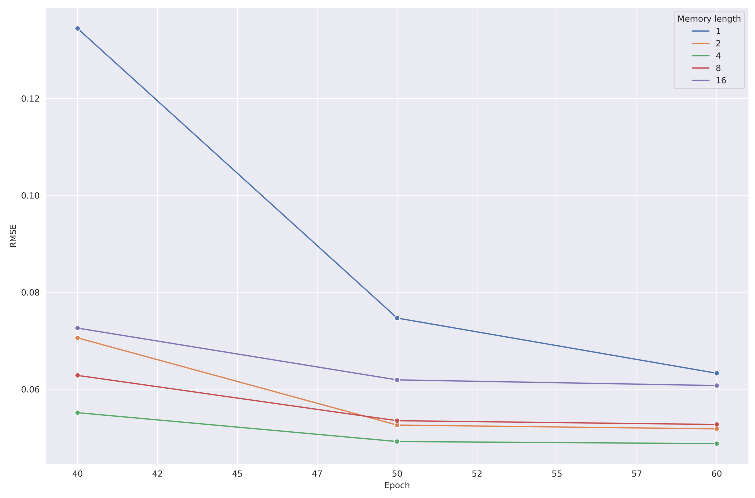


Figure 6.13: A zoomed in version of Figure 6.12 showing the performance after epoch 40 of several memory lengths plotted against the number of epochs when using a batch size of 2.

The outcome of the grid-search optimization is shown in Table 6.2. The average RMSE with their standard deviation is computed on the training and validation set. When the training error is significantly lower or higher than the validation error, it needs to be analyzed whether the model overfits or underfits the training data. It can also be used to see whether the samples contained in the training set are comparable to that of the validation set.

Table 6.2 verifies that the different configurations have a large impact on models' performance. The difference between the maximum error 0.617 and the minimum 0.039 is very significant. The same holds for the difference on the training set. Interestingly, there is no specific value for the hyperparameters that always outperforms the other values because a different mix of values is shown in the table's top rows. However, from the worst performing sets of parameters (bottom rows of the table), we can conclude that using an LSTM with a larger number of *units* outperforms an LSTM with a smaller number of *units*.

The average training error is for every configuration higher than the average validation error. This is probably due to the size of the training set. As explained earlier, the size of the training set is significantly larger than the size of the validation set. Thus, there is a higher chance that the training set contains more unpredictable events. Because of these events, it is hard for the model to learn these patterns, especially when the events occur rarely. Therefore, the model makes more errors while training and thus resulting in a higher training error. The errors of the different sets can only be compared to each other because the standard deviations are roughly the same.

The unseen test set will be used to evaluate the performance of the model when the optimal set of parameters (shown in the first row of Table 6.2) is used. The result of this will be presented in Section 6.3.

Table 6.2: Result of the grid-search optimization of the LSTM.

						Training error		Validation error	
Layers	Units	Learning rate	Dropout	Activation function	Optimizer	Mean	Std	Mean	Std
4	1024	0.01	0.0	tanh	Nadam	0.053	0.029	0.039	0.032
4	1024	0.01	0.1	relu	Adam	0.063	0.031	0.039	0.033
4	1024	0.01	0.0	tanh	Adam	0.056	0.032	0.039	0.032
4	1024	0.01	0.0	relu	Nadam	0.055	0.029	0.039	0.035
8	64	0.01	0.0	tanh	Nadam	0.062	0.037	0.039	0.034
4	256	0.01	0.0	tanh	Nadam	0.057	0.035	0.040	0.036
4	1024	0.01	0.1	tanh	Nadam	0.057	0.028	0.040	0.030
8	1024	0.01	0.1	relu	Adam	0.071	0.032	0.040	0.030
4	256	0.01	0.1	tanh	Nadam	0.070	0.033	0.040	0.035
8	1024	0.10	0.1	tanh	Adam	0.058	0.029	0.040	0.028
..	..	..	..	..	..	..	..	..	..
8	64	0.10	0.1	relu	Nadam	0.112	0.033	0.054	0.029
8	64	0.01	0.1	relu	Nadam	0.164	0.043	0.069	0.041
4	64	0.01	0.1	relu	Adam	0.450	0.125	0.431	0.138
8	64	0.01	0.1	relu	Adam	0.475	0.149	0.441	0.170
4	64	0.01	0.0	relu	Adam	0.531	0.136	0.518	0.146
8	64	0.01	0.0	relu	Adam	0.627	0.134	0.617	0.165

6.2.3. ConvLSTM

The difficulties encountered during the performance analysis of the ConvLSTM will be discussed in this section. Besides that, the results of the hyperparameter optimization will be presented.

6.2.3.1 Challenges

The number of experiments required for the optimization of the hyperparameters of the LSTM was infeasible, as explained in Section 6.2.2.2. Because the entire network structure is used as input by the ConvLSTM, the experiment does not have to be repeated for each link in the network. Therefore, the number of experiments that need to be run automatically decreases by a factor of 756 (number of links in the network). However, the number of experiments remains infeasible because of the large number of tunable hyperparameters and the high computational complexity of the ConvLSTM, which results in long runtimes. For this reason, the number of experiments is reduced by using the approach discussed in Section 6.2.2.1.

6.2.3.2 Hyperparameter Optimization

Similarly, as the optimization of the parameters of the LSTM, the result of the hyperparameter optimization of the ConvLSTM is discussed in this section. First, the impact of the number of epochs together with the batch size will be analyzed. For the number of epochs, the following values are used {5, 10, 15, 20} and for the batch size {2, 4, 8, 16, 32}. As can be noticed, the number of epochs is significantly smaller than the number of epochs used by the LSTM. This is mainly because it was found that running the model with that many epochs does result in extremely large runtimes and an overfitting model. Also, the batch size is limited to 32 because it was proven that the model performs comparably for larger batch sizes. Later in this section, an in-depth analysis will be given that verifies the above-defined statements. The experiment is run for memory lengths {1, 2, 4, 8, 16} and the validation set is used to evaluate the performance of the trained model.

Figure 6.14 contains the result of the aforementioned experiment. Each cell value in the heatmap shows the average performance of all memory lengths using the defined number of epochs and batch size. Interestingly, when the batch size is larger than 8, the number of epochs has a negligible impact on the model's performance. Because of this result, the maximum batch size was limited to 32. On the other hand, the amount of epochs does have a significant impact on the performance for smaller batch sizes, 2, 4, and 8, respectively. Limiting the number of epochs to 20 can be one of the reasons why the number of epochs has little impact on the performance when using a large batch size. This indicates that the model needs more epochs to learn when using a large batch size, just like the LSTM model. To verify this, the number of epochs needs to be increased. Because this also increases the runtime of the algorithm, it has been decided to leave this as future work. This is only possible because the other configurations perform significantly better without optimizing any other hyperparameters. The configuration that uses a batch size of 2 and 15 epochs outperforms the other configurations and results in an average RMSE of 0.040. It is worth noting that when using 20 epochs, the RMSE increases by 0.001 when a batch size of 2 is used. Because the cell values in the heatmap contain the average performance of the different memory lengths, a separate plot will be created to see if this statement holds for every l. When this holds for the best performing l, the number of epochs should be limited to 15.

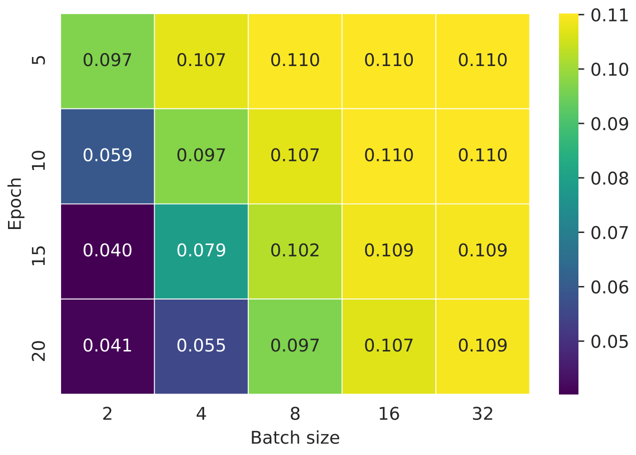


Figure 6.14: Heatmap showing the relation between the performance of the ConvLSTM and the hyperparameters epochs and batch size. The color of the cell represents the RMSE, thus a darker value indicates better performance.

The impact of the memory length on the performance has not been analyzed yet. To see how the memory length impacts the performance, the performance is plotted against the number of epochs for each memory length when using a batch size of 2. In general, smaller batch sizes combined with a large number of epochs gave the best results, as shown in Figure 6.14.

Figure 6.15 shows that this holds for memory lengths 8 and 16 because the RMSE only decreases. However, for memory lengths 1, 2, and 4, the RMSE increases when the number of epochs is larger than 15. This can be explained by the fact that after epoch 15, the model starts to overfit the training data and fails to generalize. For this reason, the number of epochs used in the grid-search optimization is limited to 15. It is also clear that smaller l yields better performance. Using a memory length of 1 seems like the best option because it significantly outperforms the other ls. The impact of the memory length on the performance is significantly larger than the impact of the memory length on the performance of the LSTM. This could be a result of the small number of epochs used because we see that the RMSE still decreases for $l = 8$ and $l = 16$. This seems to indicate that the performance of the models using a large memory length can be improved if the number of epochs is increased. Another interesting insight is that the models using smaller l learn significantly faster than larger l because the RMSE decreases faster. For the LSTM, the opposite was true; the LSTM models that use larger l learn faster than those that use smaller l as shown in Figure 6.12. This is probably due to the difference in the input. The ConvLSTM takes several graph snapshots as input. When using a large memory length, a large number of graph snapshots are used to learn. Because of this large input, the model needs more time to learn the patterns and filter out noise. This is most likely why the ConvLSTM needs more epochs to learn than when a smaller memory length is used. The results are shown in Figure 6.14 and Figure 6.15 have shown that it is a proper choice to run the grid-search optimization with a batch size of 2, memory length of 1 and 15 epochs.

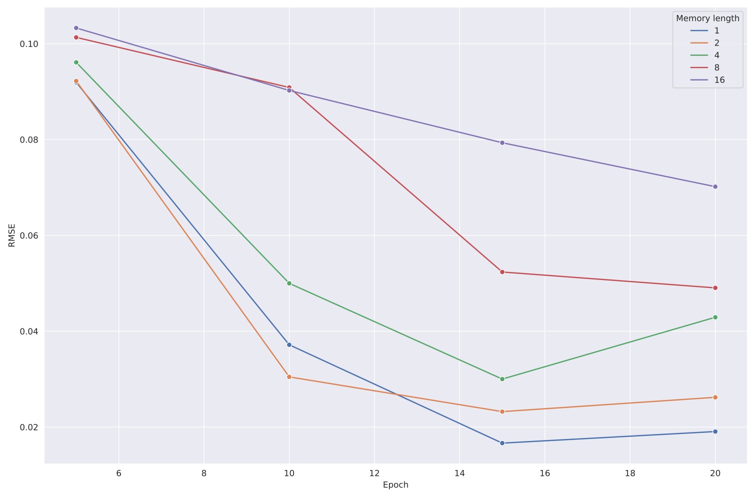


Figure 6.15: The performance of several memory lengths plotted against the number of epochs when using a batch size of 2.

Table 6.3 contains the result of the grid-search optimization. When looking at the average validation error of the different configurations, it can be seen that the configuration of the ConvLSTM has a large impact on the performance. The difference between the best performing and worst performing set of parameters is 0.072 - 0.014 = 0.058. This highlights the importance of the optimization of the hyperparameters. From the first row of the table, we can see that we reach an RMSE of 0.014, which is an extremely good performance. Besides that, the standard deviation is small, indicating that the performance of the models on the different folds created with the cross-validation technique is almost identical. It is also interesting to see that the average training error is comparable to the validation error for all configurations. The same holds for the standard deviation. This indicates that the model generalizes because it also performs comparable on the unseen (validation) set. When looking at the top rows of the table, we can see that the performances are comparable, but the parameters used in the configuration are relatively different. For example, for the parameters *filters*, *kernel size* and *optimizer* we do not have a value that is preferred in all configurations. Looking at the button rows, it can be noticed that the *learning rate* is always 0.01, and the *optimizer* is equal to the Adam optimizer. This indicates that for this dataset, the combination of a small *learning rate* and the Adam algorithm as *optimizer* does not perform well.

Table 6.3: Result of the grid-search optimization of the ConvLSTM.

					Training error		Validation error	
Filters	Kernel size	Learning rate	Activation function	Optimizer	Mean	Std	Mean	Std
20	7	0.10	relu	Nadam	0.013	0.001	0.014	0.001
40	5	0.10	relu	Nadam	0.015	0.009	0.016	0.008
40	7	0.10	relu	Adam	0.021	0.019	0.022	0.019
40	5	0.10	relu	Adam	0.021	0.016	0.022	0.016
20	7	0.10	relu	Adam	0.022	0.017	0.022	0.017
40	7	0.01	relu	Nadam	0.022	0.009	0.022	0.009
40	7	0.10	relu	Nadam	0.023	0.020	0.023	0.020
20	7	0.01	relu	Nadam	0.023	0.007	0.023	0.007
20	5	0.10	relu	Adam	0.024	0.020	0.024	0.020
20	3	0.10	relu	Nadam	0.024	0.017	0.024	0.016
..	..	..	..	..	..	..	..	..
40	3	0.01	tanh	Adam	0.065	0.003	0.064	0.003
20	7	0.01	tanh	Adam	0.065	0.002	0.065	0.002
40	3	0.01	relu	Adam	0.066	0.018	0.065	0.018
20	5	0.01	tanh	Adam	0.069	0.003	0.068	0.003
20	3	0.01	tanh	Adam	0.072	0.004	0.071	0.003

The parameters defined in the first row of the table will be used to train the model, and the test set will be used to evaluate the models' performance. The configuration of the top row of the table is chosen because it performs optimally on the training set as well as the validation set, indicating that using this set on the test set will give the best results. The result of this will be presented in Section 6.3.

6.2.4. Summary

This section showed why it is important to optimize the hyperparameters of a model. Besides that, we proved that predicting the demand for the next day does not bring any challenges and does not add any value for the company because it predicts the same value as the input value used for the prediction. This was verified by analyzing the structure of the transition matrix of the Markov chain. By using a white-box model such as the Markov chain, the reason why a model predicts a specific value can be explained easily. This emphasizes why it is important to start with a white-box baseline method to understand the prediction behavior of the model. Reflecting on research subquestion A4 (see Section 1.3), "What is the impact of the number of time steps k used to predict on the performance of the model?", a few insights have been gained. As shown in this section, the number of time steps used, just like the other hyperparameters, is an important hyperparameter that needs to be fine-tuned for each model. For the Markov chain, a memory length of 2 results in the best performance, while for the LSTM, a memory length of 4 works best, and for the ConvLSTM, a memory length of 1. This can be explained by the fundamental difference in how the different models learn and predict, also relating to the other hyperparameters used in the algorithms. Note that the number of data points used by each model is relatively small. This indicates that the more historical data points used to predict, the harder it is for the models to predict accurately, indicating that data does not contain many long-term patterns.

In Section 6.1.3, we already showed how to quantify the performances of the different models systematically. Reflecting on research subquestion A3, "What is a suitable metric to evaluate the predictions made by the models?" we can conclude that different metrics can be used to evaluate the performance of the models. Both the MAE and RMSE will be used to evaluate the performances of the models. Also, the accuracy is calculated, indicating the number of correct predictions divided by the total predictions. Note that the predicted values are first discretized to the closest original value of the dataset. The average accuracy can be compared to the theoretical upper bound of the predictability of the network. In this way, we can determine whether any improvements can be made.

6.3. Performance Comparison

In this section, the performances of the three implemented models on the test set will be compared with each other. Also, the accuracy of the models will be compared to the upper bound of the predictability of the network. Lastly, the predictions will be visualized to compare the prediction behavior of the different models.

6.3.1. Evaluation of the Performance

In this section, the final performance of the different models is discussed. All models are trained on the same training set and are evaluated on the same test set to compare the performance of the models systematically. As already briefly highlighted in Section 6.1.3.2, the theoretical upper bound needs to be compared to the accuracy of the models. To compute the accuracy, the predicted values are discretized to the closest value in the dataset used for the predictions. Because the Markov chain and LSTM use the information of a single link to predict, only the unique values of that link will be used when discretizing the predictions. However, the ConvLSTM uses the information of the entire network, and therefore, the predictions are discretized to the closest value observed in the entire network. To analyze the impact of the discretization of the predictions, the discretized predictions and the predicted values are compared to the actual values. In this way, we can see whether the discretization improves the performance of the models and the impact of the discretization for the different models.

So far, the RMSE is used to evaluate the performances of the models. It is also interesting to compute the MAE and compare the models' performances using this evaluation metric. Section 6.1.3.1 explains the difference between these two evaluation metrics and why the RMSE penalizes larger errors more. When the RMSE is significantly larger than the MAE, it could tell something about the number of large errors made.

It is important to note that when evaluating the performance of the ConvLSTM, an interesting finding was made: the RMSE and MAE were both exceptionally low. When investigating why this was the case, it was discovered that the sparsity of the adjacency matrix has a significant impact on the performance of the ConvLSTM. As a result, while evaluating the performance of the ConvLSTM, cells in the adjacency matrix that remain 0 all of the time are excluded from the evaluation. Cells in the adjacency matrix that remain 0 all of the time represent a specific type of units that do not exist at a specific storage facility. Therefore, it is reasonable to exclude them when evaluating. In this way, the performances can be fairly compared with each other.

Table 6.4 contains the final performance of all models. The model that performs optimally is highlighted for each metric. The LSTM outperforms the other models for all metrics. Next, we will try to explain why the LSTM outperforms the other models. If we compare the performance of the LSTM, which only uses the information of a single link, with the performance of the ConvLSTM, which uses the information of the entire network. We see that the LSTM outperforms the ConvLSTM for each metric. This indicates that using an input with an extra dimension does not positively impact the performance. Section 4.3.3 proves that the average cross-correlation between the links is relatively low. Besides that, the average standard deviation was relatively high. The high standard deviation indicates that there are some links in the network which have some correlation. The ConvLSTM uses the (low) correlation between the links to learn and predict. Because of this low correlation, the model tries to capture the spatio-temporal dependencies, which generally does not contain useful prediction information. Thus, more noise is introduced by using this information. This is why the LSTM outperforms the ConvLSTM because the LSTM only uses the information of a single link that has a high autocorrelation, making it easier for the model to learn and results, therefore in better performance.

Table 6.4: Final performances of the different models.

	MAE	MAE (discretized)	RMSE	RMSE (discretized)	Accuracy
Markov chain	0.061	0.055	0.092	0.091	0.317
LSTM	**0.048**	**0.044**	**0.070**	**0.069**	**0.355**
ConvLSTM	0.061	0.058	0.094	0.093	0.130

The Markov chain and LSTM both use only the temporal information of a single link to predict. Although both models use the same data, the LSTM outperforms the Markov chain model for all metrics. This shows that for this dataset, a more complex model is more suitable. When comparing the MAE, we see that the difference between the models is equal to 0.061 - 0.048 = 0.013. For the RMSE, the difference is equal to 0.092 - 0.070 = 0.022, and thus slightly larger. Comparing the MAE with the RMSE, we see that the difference between these two metrics is not extremely large. This is as expected because for this dataset, making huge errors is not that likely. This can be explained by the fact that the weekly demand does not fluctuate violently because this would indicate that a large fraction of the customers who rent a unit with the same volume, floor, and location start or stop renting in the same week. Section 4.3.2 proved that the temporal data of a single link has a high autocorrelation at the first few lags. It is worth mentioning that only the daily dataset's autocorrelation was calculated. However, evaluating the performance of the models on the weekly dataset reveals that the weekly data is still relatively highly correlated, as the models perform reasonably.

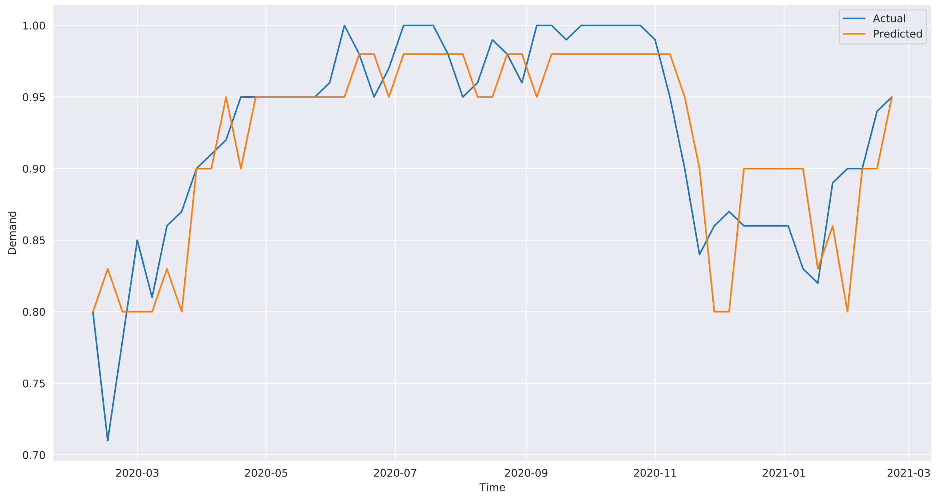
When comparing the training and validation error discussed in Section 6.2 with the test error, it was found that for all models, both the training error and validation error are lower than the test error. For the Markov chain, the difference between the training and test errors was equal to 0.092 - 0.046 = 0.046, and for the LSTM, the difference was equal to 0.070 - 0.053 = 0.017. Having a lower training error is as expected because the data used to train the model is used to evaluate the model. The model does not overfit because we showed that the training error is comparable to the validation error. By comparing the difference between the training and test set for both models, we can conclude that the LSTM fits better to unseen data because the difference between the training and test set is significantly smaller.

The discretization of the predictions has a positive impact on the performance of all models. This holds because the MAE (discretized) is always smaller than the MAE and the same is true for the RMSE (discretized) and RMSE. However, the differences are not significant. This shows that the models predict values close to or equal to the actual values in the dataset. Since the performance increases, it is recommended to always discretize the predicted values to the closest value of the dataset. It is interesting to see that the discretization of the ConvLSTM is comparable to the impact of the discretization of the Markov chain and LSTM, while the ConvLSTM uses the values of the entire network to discretize. From this, we can conclude that the discretization using the entire network's data does not have a large impact on the performance.

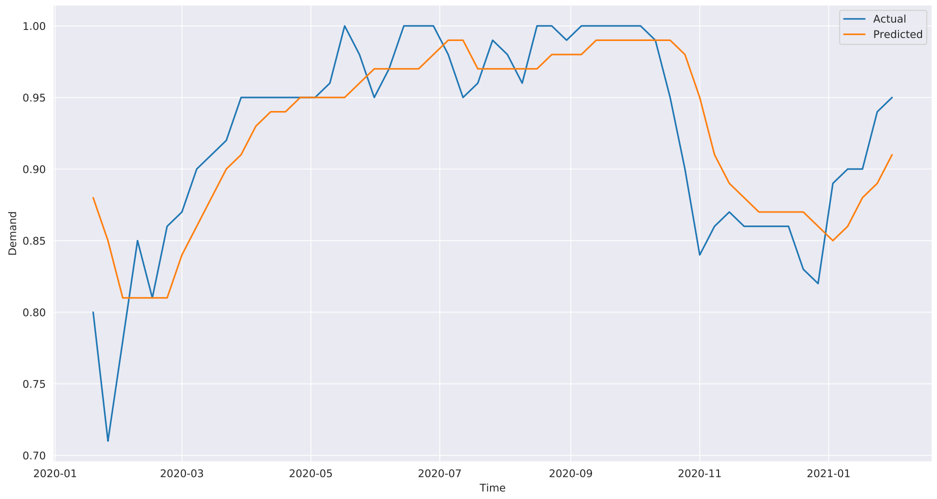
The last metric not discussed yet is accuracy. Again, the LSTM performs optimally. The difference between the accuracy of the Markov chain and LSTM is not extremely large. For the ConvLSTM, the opposite is true. Both the Markov chain and LSTM outperform the ConvLSTM by a factor of more than 2 when comparing the accuracies. The accuracy of the models is compared to the theoretical upper bound of the predictability of the network to see how well the models perform. The theoretical upper bound for this network is equal to 0.815. This is a theoretical upper bound, and no predictive algorithm can predict an accuracy above 0.815. This result indicates that there is still room for improvement because the accuracy of the implemented models is not even close to the theoretical upper bound. Why the accuracy of the models is low compared to the theoretical upper bound is an interesting question. One reason for this can be the limited amount of data to learn from. In total, the dataset consists of around 210 data points. A fraction of 70% is used for training while the other 30% is used for testing. This means that all models use approximately 150 data points to learn. This number is relatively small, especially when compared to the sizes of the datasets used in other projects. Another reason could be the selected models. Many different techniques can be used to predict, and it is impossible to implement them all because of limited time. By selecting the three implemented models, we analyzed the fundamental difference of the approaches and the impact on how the problem is modeled. In Section 7.2, it will be briefly discussed which models might be interesting to implement when this project is continued.

6.3.2. Visualization of the Performance

In this section, the predictions will be visualized. This indicates what the prediction behavior is of the different models. The predicted demand is plotted for a randomly selected UnitLTG. Because the Markov chain and LSTM both outperform the ConvLSTM, the predictions are only plotted for the first two mentioned models in Figure 6.16. From the figure, we can see that the Markov chain can only predict a discrete set of values, while the LSTM does not have this restriction, and therefore, can predict a much more smoothed line. It is interesting to see that the predictions are relatively close to the actual values for both models. We can see that the models follow the general trend with a slight delay. Besides that, both models cannot correctly predict when the demand starts to increase or decrease. This shows that the dataset does not contain many recurring patterns. This could also be due to the relatively small size of the training data.



(a) Predictions of the Markov chain.



(b) Predictions of the LSTM.

Figure 6.16: Predictions of the Markov chain and LSTM visualized. The demand is plotted for the units with volume 1m^3 at the ground floor in Leiderdorp.

6.3.3. Summary

Reflecting on research subquestion A2 (see Section 1.3), "Which type of technique is most suited for the demand prediction problem, and why?" and subquestion A2.1 "Do black-box methods outperform white-box methods?", a few insights have been gained. First of all, we can conclude that the LSTM outperforms the other two implemented methods. The ConvLSTM can not outperform the LSTM because of the low correlation between the links. Because of the low cross-correlation, the ConvLSTM can not capture valuable spatio-temporal dependencies that the ConvLSTM uses to learn and predict. The LSTM slightly outperforms the Markov chain model, and both use only the information of a single link with a high autocorrelation. Because of this high correlation, the models can predict the demand with a relatively low error. This emphasizes the importance of analyzing the data because this can explain why a particular model performs better than other models. This section also shows that black-box methods do not always outperform white-box methods because the ConvLSTM cannot outperform the Markov chain model.

7 Conclusion & Future Work

In this chapter, the work of this thesis will be summarized, and conclusions will be drawn. In addition, future directions for this research will be suggested.

7.1. Conclusions

In this project, we tried to investigate whether it is possible to forecast the demand of the units. The Markov chain, LSTM, and the ConvLSTM were selected and applied to solve the prediction problem. Using these three predictive algorithms, we analyzed whether utilizing the data of the entire network provides any benefits compared to utilizing only the information of a single link to predict. In addition, it is researched if the black-box models outperform the white-box model. The following conclusions can be drawn from the data analysis, discussed in Chapter 4, and the results of the experiments discussed in Chapter 6:

- From the data analysis, some interesting insights were gathered. First of all, it was shown that the volume, floor, and location all impact the distribution of the demand. After computing the autocorrelation between the time series and a lagged version of the time series itself, we showed that there exists a high correlation between consecutive data points. This result confirms that it is possible to predict the demand for the next step based on historical data. Finally, the cross-correlation function was used to compute how correlated different time series are. The result showed that different time series have a low cross-correlation, indicating that the demand of the storage units of each location, floor, and volume combination changes uniquely over time. This verifies that the temporal change of the units is non-trivial, which makes it an interesting problem to research.

- We have proven that the daily data does not fluctuate frequently. We showed that in more than 90% of the time, the forecasted demand value was the same as the previous one. Because of the few fluctuations, a weekly dataset was created to make the problem more challenging and make it more interesting from a business perspective due to the higher prediction timeframe. This dataset is used as input by the models. By comparing the performance of the models on the daily dataset with the performance on the weekly dataset, it was verified that the weekly dataset was harder to predict because of more fluctuating, and thus, larger errors.

- After comparing the performance of the three state-of-the-art predictive algorithms, we observed that the LSTM outperforms the other models in terms of MAE, RMSE, and accuracy. The LSTM outperforms the ConvLSTM, proving that utilizing information of multiple links in the network, instead of utilizing the information of a single link, does not have a positive impact on the performance. This is due the low correlation between the links in the network, as proven by the data analysis. The Markov chain performs comparable to the ConvLSTM. This highlights that a complex black-box model does not always outperform a white-box model. This proves the importance of data analysis because it can be used to explain why the ConvLSTM is not able to outperform the Markov chain. For this dataset, a more complex model using only the information of a single link to predict is more suitable because the LSTM outperforms the Markov chain. Furthermore, the difference between the training and test error is minimal for the LSTM, indicating that the LSTM is more suitable to adapt to unseen data. Besides, because the errors between the training and test set are small for each model, it is also proven that the models do not overfit the training data. Finally, we observed that the prediction errors in each model are caused mainly by the inability to predict a change in the demand correctly.

- We also verified that the hyperparameters used by the models have a significant impact on the performance of the models, emphasizing the importance of hyperparameter optimization. We can also conclude that the optimal memory length depends on the fundamentals of a model and is not equal for each model.
- When we compare the accuracy of the applied algorithms to the theoretical upper bound of the predictability of the network, we can conclude that the algorithms do not reach an accuracy close to the upper bound. This highlights that there is still room for improvement. In Section 7.2, a list of potential directions is given how to reach a higher accuracy.

7.2. Future Work

Because of the limited time window of this project, there are several research topics not investigated yet. Therefore, in this section, a list of potential directions is given to investigate further when the thesis project is continued. The following recommendations are:

- In Chapter 4 it is proven that the demand of each link in the network changes uniquely over time because of the low cross-correlation. Therefore, the LSTM performs optimally when the hyperparameters are optimized for each link separately. As discussed in this thesis, only a subset of time series is used to find the optimal hyperparameters. Therefore, it is interesting to investigate if the performance of the LSTM increases when the LSTM is trained for each link separately. In particular, it is interesting to see whether we can reach a higher accuracy closer to the theoretical upper bound.
- Only three state-of-the-art predictive algorithms were implemented. Because of the relatively low accuracy, different models can be applied to see how these models perform compared to the Markov chain, LSTM, and ConvLSTM. It is recommended to apply models that only use a single link's information to predict because of the low correlation between the links. Besides using a different algorithm to predict, it is also possible to apply a stacked algorithm. A stacked algorithm uses the output of several different predictive algorithms as input and tries to utilize the strength of each algorithm.
- In this project, only the last four years of data are used for the data analysis and training of the models. It is worth analyzing whether using more years of data positively impacts the performance of the models. However, it needs to be paid attention that this larger dataset remains structured and consistent and requires more computational expenses to train the models.
- The use of additional datasets is another interesting idea that could help improve the performance of the models. There might be some interesting datasets, such as the number of telephone calls to a storage facility or the number of website visitors, which might correlate with demand. The website visitors can be linked to a storage facility because you need to select the storage facility when you visit the company's main page. Because this information can not be directly linked to a specific type of unit, it does not create additional features for the edges in the network. However, it does create additional features for the nodes (locations) on the left-side of the network.
- In this project, the focus was on predicting the demand for existing nodes using only three input variables. However, another interesting research question is about whether we can predict the demand for newly added nodes. This is important when the company decides to expand an existing storage facility by adding a layer on the existing floors. When this happens, the optimal mixture of units needs to be determined. Thus, the number of units for each volume needs to be defined. When it is possible to predict the demand for newly added nodes representing the units on the new floor, the company can use this information to determine the optimal mixture of units. This problem has some similarities with the cold start problem in recommendation systems. The problem with these recommendation systems is that they do not have any information on newly added users or items, making it hard for the system to recommend.

A
Implementation Details

A.1. Markov Chain

The pseudocode of the Markov chain is shown in Algorithm 2.

Algorithm 2: Markov Chain

```
Result: Performance of the model
time_series = get time series data and split into $n$ - $l$ sequences, where $n$ is the length of the time series
unique_states = list containing the unique combinations of the time_series of $l$
transition_matrix = m × m matrix (|m| = length of unique_states) initialized with zeros
train, test = split time_series into train and test set
for sequence in train do
    input = sequence[0:$l$ - 1]
    output = sequence[1:$l$]
    row_index_matrix = find row index of transition_matrix for input
    col_index_matrix = find column index of transition_matrix for output
    transition_matrix[row_index_matrix, col_index_matrix] += 1
end
transition_matrix = divide each row of the transition_matrix by the sum of that row to ensure that the sum of each row
  is 1
actual_values = []
predicted_values = []
for sequence in test do
    input = sequence[0:$l$ - 1]
    row_index_matrix = find row index of transition_matrix for input
    predicted_value = find column name containing highest cell value for row row_index_matrix in transition_matrix
    add predicted_value to predicted_values
    actual_value = last element of sequence
    add actual_value to actual_values
end
errors = apply evaluation metric on actual_values and predicted_values
return errors
```

B
Extended Data Analysis

B.1. Distribution

B.1.1. Floors

In Figure B.1, the demand for two different unit volumes (rows) and three different floors (columns) is shown. From this plot, it can be concluded that the floor on which the units are located impact the demand distribution. Therefore, the floor at which the units are located should be taken into account when predicting.

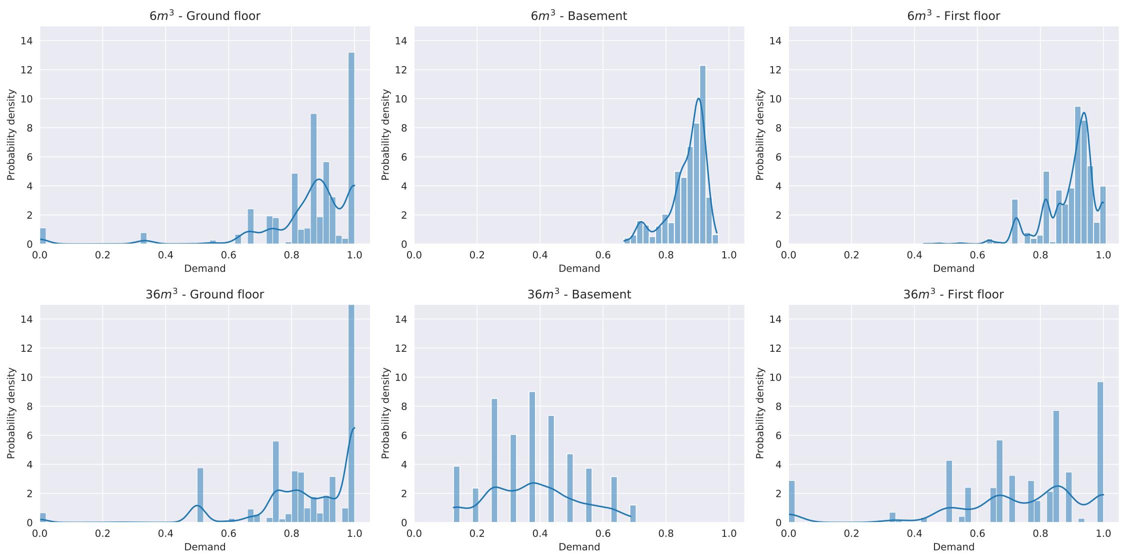


Figure B.1: Estimated probability density function for the combined demand for two unit volumes taking the floor into account.

B.1.2. Locations

In Figure B.2, the demand distribution of nine different locations is shown. From this plot, it is immediately clear that the location has an impact on the distribution. For example, the demand within Amsterdam-Duivendrecht, Tilburg, Rotterdam, s-Hertogenbosch, and Amsterdam are highly centered around 0.8 or higher. On the contrary, for locations Leiderdorp and Almere, the mean is found between 0.5 and 0.6. This result shows that the location is of importance when making the predictions.

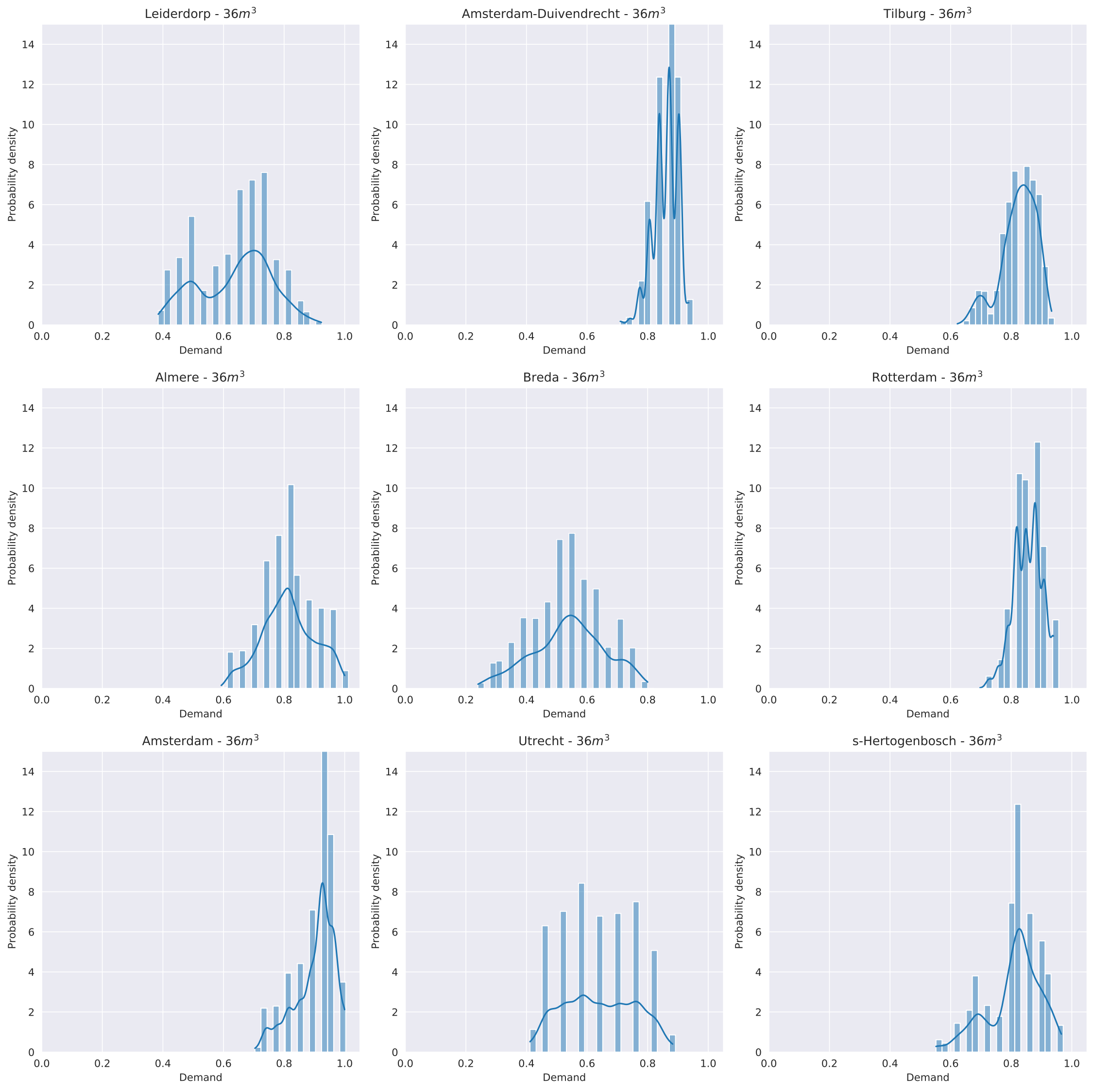


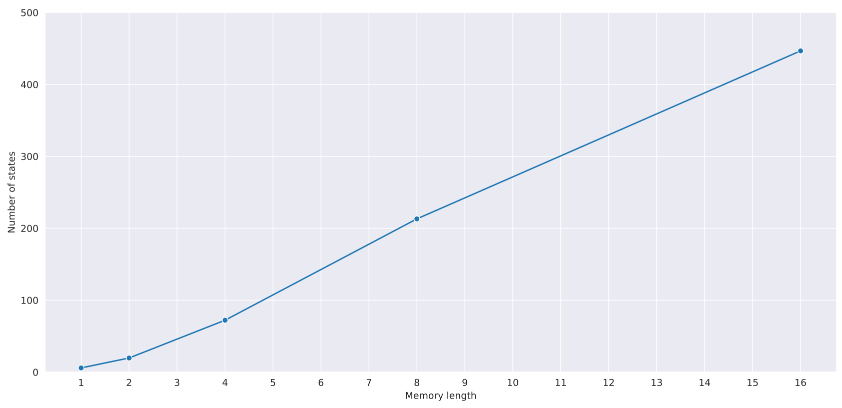
Figure B.2: Estimated probability density function for unit volume 36m^3 for several locations.

C
Extended Experimental Results

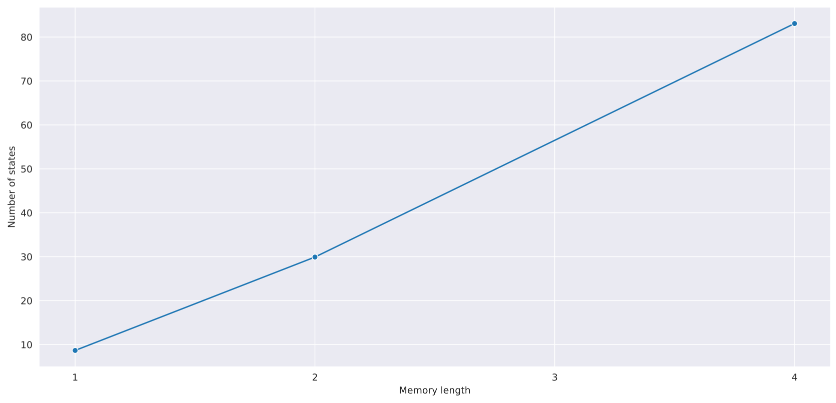
C.1. Markov Chain

C.1.1. Number of states

In Figure C.1, the average number of states is plotted against the memory length. Figure C.1a shows the average number of states when using the daily data, and Figure C.1b shows the average number of states when using the weekly dataset. From these plots, we can conclude that the number of states increases linearly with the memory length. It also highlights that the memory length should be kept small because of the relatively small size of the dataset and the large number of states when using a larger memory length.



(a) Average number of states of the Markov chain based on the daily dataset. chain.



(b) Average number of states of the Markov chain based on the weekly dataset.

Figure C.1: The average number of states plotted against the memory length.

C.1.2. Number of reachable states

In Figure C.2, the average number of reachable states is plotted against the memory length based on the weekly dataset. This plot shows that the larger the memory length, the lower the number of reachable states. This is due to the large number of unique states when using a larger memory length.

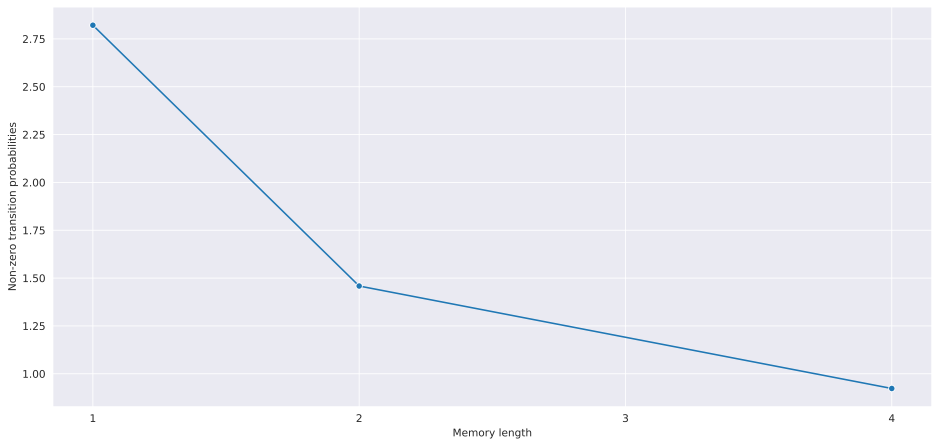


Figure C.2: Average number of reachable states plotted against the memory length based on the weekly dataset.

Bibliography

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[2] D. Tang, W. Du, L. Shekhtman, Y. Wang, S. Havlin, X. Cao, and G. Yan, "Predictability of real temporal networks," *National Science Review*, vol. 7, no. 5, pp. 929–937, 2020.

[3] P. Holme and J. Saramäki, "Temporal networks," *Physics reports*, vol. 519, no. 3, pp. 97–125, 2012.

[4] X.-X. Zhan, A. Hanjalic, and H. Wang, "Information diffusion backbones in temporal networks," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[5] L. J. Peters, J.-J. Cai, and H. Wang, "Characterizing temporal bipartite networks-sequential-versus cross-tasking," in *International Conference on Complex Networks and their Applications*, pp. 28–39, Springer, 2018.

[6] A. Lancichinetti and S. Fortunato, "Consensus clustering in complex networks," *Scientific reports*, vol. 2, no. 1, pp. 1–7, 2012.

[7] B. Kaya and M. Poyraz, "Age-series based link prediction in evolving disease networks," *Computers in biology and medicine*, vol. 63, pp. 1–10, 2015.

[8] M. S. Lau, G. Marion, G. Streftaris, and G. J. Gibson, "New model diagnostics for spatio-temporal systems in epidemiology and ecology," *Journal of The Royal Society Interface*, vol. 11, no. 93, p. 20131093, 2014.

[9] N. Benchettara, R. Kanawati, and C. Rouveirol, "A supervised machine learning link prediction approach for academic collaboration recommendation," in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 253–256, 2010.

[10] L. Yao, L. Wang, L. Pan, and K. Yao, "Link prediction based on common-neighbors for dynamic social network," *Procedia Computer Science*, vol. 83, pp. 82–89, 2016.

[11] A. MySQL, "Mysql," 2001.

[12] J. G. Kemeny and J. L. Snell, *Markov chains*, vol. 6. Springer-Verlag, New York, 1976.

[13] T. M. Rutkowski, R. Zdunek, and A. Cichocki, "Multichannel eeg brain activity pattern analysis in time–frequency domain with nonnegative matrix factorization support," in *International Congress Series*, vol. 1301, pp. 266–269, Elsevier, 2007.

[14] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[15] A. Krohn-Grimberghe, L. Drumond, C. Freudenthaler, and L. Schmidt-Thieme, "Multi-relational matrix factorization using bayesian personalized ranking for social network data," in *Proceedings of the fifth ACM international conference on Web search and data mining*, pp. 173–182, 2012.

[16] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, pp. 1–27, 2011.

[17] S. Gao, L. Denoyer, and P. Gallinari, "Temporal link prediction by integrating content and structure information," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1169–1174, 2011.

[18] W. Yu, C. C. Aggarwal, and W. Wang, "Temporally factorized network modeling for evolutionary network analysis," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 455–464, 2017.

[19] X. Ma, P. Sun, and Y. Wang, "Graph regularized nonnegative matrix factorization for temporal link prediction in dynamic networks," *Physica A: Statistical mechanics and its applications*, vol. 496, pp. 121–136, 2018.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] S. Siami-Namini and A. S. Namin, "Forecasting economics and financial time series: Arima vs. lstm," *arXiv preprint arXiv:1803.06386*, 2018.

[22] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 4, pp. 694–707, 2016.

[23] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, “Convolutional, long short-term memory, fully connected deep neural networks,” in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4580–4584, IEEE, 2015.

[24] D. Wang, Y. Yang, and S. Ning, “Deepstcl: A deep spatio-temporal convlstm for travel demand prediction,” in *2018 international joint conference on neural networks (IJCNN)*, pp. 1–8, IEEE, 2018.

[25] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[26] H. Wang and B. Raj, “On the origin of deep learning,” *arXiv preprint arXiv:1702.07800*, 2017.

[27] L. Xu, J. S. Ren, C. Liu, and J. Jia, “Deep convolutional neural network for image deconvolution,” *Advances in neural information processing systems*, vol. 27, pp. 1790–1798, 2014.

[28] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[29] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, “T-gcn: A temporal graph convolutional network for traffic prediction,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3848–3858, 2019.

[30] L. Ge, H. Li, J. Liu, and A. Zhou, “Temporal graph convolutional networks for traffic speed prediction considering external factors,” in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pp. 234–242, IEEE, 2019.

[31] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, “Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 388–396, IEEE, 2019.

[32] M. Gao, L. Chen, B. Li, Y. Li, W. Liu, and Y.-c. Xu, “Projection-based link prediction in a bipartite network,” *Information Sciences*, vol. 376, pp. 158–171, 2017.

[33] S. Aslan and B. Kaya, “Time-aware link prediction based on strengthened projection in bipartite networks,” *Information Sciences*, vol. 506, pp. 217–233, 2020.

[34] T. Wu, S.-H. Yu, W. Liao, and C.-S. Chang, “Temporal bipartite projection and link prediction for online social networks,” in *2014 IEEE International Conference on Big Data (Big Data)*, pp. 52–59, IEEE, 2014.

[35] O. Kart, O. Ulucay, B. Bingol, and Z. Isik, “A machine learning-based recommendation model for bipartite networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 553, p. 124287, 2020.

[36] X. Cai, J. Shang, Z. Jin, F. Liu, B. Qiang, W. Xie, and L. Zhao, “Dbge: employee turnover prediction based on dynamic bipartite graph embedding,” *IEEE Access*, vol. 8, pp. 10390–10402, 2020.

[37] İ. İşlek and Ş. G. Öğüdücü, “A retail demand forecasting model based on data mining techniques,” in *2015 IEEE 24th International Symposium on Industrial Electronics (ISIE)*, pp. 55–60, IEEE, 2015.

[38] R. Jin, T. Xia, X. Liu, T. Murata, and K.-S. Kim, “Predicting emergency medical service demand with bipartite graph convolutional networks,” *IEEE Access*, vol. 9, pp. 9903–9915, 2021.

[39] F. Mutinda, A. Nakashima, K. Takeuchi, Y. Sasaki, and M. Onizuka, “Time series link prediction using nmf,” *Journal of Information Processing*, vol. 27, pp. 752–761, 2019.

[40] F. J. Montáns, F. Chinesta, R. Gómez-Bombarelli, and J. N. Kutz, “Data-driven modeling and learning in science and engineering,” *Comptes Rendus Mécanique*, vol. 347, no. 11, pp. 845–855, 2019.

[41] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, “Addressing cold-start problem in recommendation systems,” in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pp. 208–211, 2008.

[42] D. W. Scott, “Scott’s rule,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 497–502, 2010.

[43] F. Chollet *et al.*, “Keras.” `https://keras.io`, 2015.

[44] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation.,” *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.