Analysis of the Utilization of Mobile Network Base Stations Using Traffic Load Predictions

by

Basma Mahdy

A thesis submitted to the Department of Electrical and Computer Engineering
In conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
May 2020

Abstract

Every day, mobile network traffic is increasing in an unprecedented manner all over the world, resulting in growing demand from the network operators to deploy more base stations in order to be able to serve more devices while maintaining a satisfactory level of service quality. Base stations are considered the leading energy consumer in a network infrastructure; consequently, increasing the number of base stations will increase power consumption. By finding a method to predict the traffic load on base stations, network optimization techniques can be applied to put inactive base stations into sleep mode thereby decreasing energy consumption.

This research explores methods capable of predicting traffic load on base stations. The most common time series forecasting techniques are examined in this research on a public dataset that provides records of traffic loads of several base stations over the span of one week.

Because of the limited number of records that exist in the dataset for each base station, and to avoid a common problem often raised in training forecasting algorithms, different base stations are grouped together while building the prediction model. Due to the different behavior of the base stations, forecasting the traffic load of multiple base stations together becomes challenging. Our proposed solution involves clustering the base stations according to their behavior and forecasting the load on the base stations in each cluster individually. Different clustering algorithms along with several similarity metrics are compared to each other using a variety of evaluation methods. The base stations load prediction task is formulated as a time series forecasting problem. The most common statistical techniques, machine learning techniques, and deep learning techniques used for time series forecasting are applied on the clusters. Consequently, the clusters' performances are evaluated and compared. Lastly, two popular online tools, created specifically for the time series forecasting task, are employed to the dataset. The results generated by these online tools are

used in this research as a benchmark to compare to the performance of the proposed prediction model.

Our findings demonstrate that Deep Recurrent Neural Networks such as the Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) networks perform better than other time series forecasting techniques for large data sets. Our findings offer clustering the time series data according to their behavior before using them for the forecasting algorithms as a solution for the problem of the dissimilar behavior of the time series when they are trained together.

Acknowledgments

I cannot begin to express my great appreciation to my supervisors, Prof. Hossam Hassanein and Prof. Aboelmagd Noureldin who always supported and helped me during my studies. I have been so lucky to have such supervisors who were patient, encouraging and always willing to help. I have learned a lot from them and I will always be grateful that I had them as my supervisors. This two-year journey was a life changing experience for me and it could not have happened without them.

Also, I must express my profound gratitude to Prof. Hazem Abbas without whom I would not have been able to complete this research. His continuous support and guidance are very much appreciated. He was always willing to help and his office door was always open. I have learned so much from him and I will always remember his great assistance during both my undergraduate and my master's degrees.

My grateful thanks are extended to Basia Palmer for her continuous help. She always made her best to make our lives easier. I am gratefully indebted to her for her continuous support during my masters and for her very valuable feedback on this thesis. I have learned a lot from her and I greatly appreciate her help.

I wish to acknowledge the support and unconditional love of my family, my mother, Sahar; my father, Mohamed; and my sisters, Rana and Riham. They were always providing me with unfailing support and continuous encouragement throughout all my life and I would never reach anything in life without them.

My thanks and appreciation go to my travelling companions, Habiba, Rawan, Sara, Maryam and Shaza, who were always here for me. Their encouragement and continuous support meant the world to me and I will always remember the truly beautiful moments that we shared during this journey.

Last but not least, I will always be thankful to my dearest friends, Rawan, Israa, Yomna, Salma, Passant, Tasneem and Yousra, who were always encouraging and motivating me through this journey.

Table of Contents

List of Figures

List of Tables:

List of Abbreviations:

C2TM	City Cellular Traffic Map Dataset
NaN	Not a Number
SDN-IoT	Software Defined Networking-based IoT
MANET	Mobile Ad-hoc Networks
MBSs	Macro Base Stations
PBSs	Pico Base Stations
SMA	Simple Moving Average
ARIMA	Autoregressive Integrated Moving Average
RNNs	Recurrent Neural Networks
SVR	Support Vector Regression
HSTNet	Hybrid Spatiotemporal Network
BTS	Base Transceiver Stations
BSC	Base Station Controller
SGSNs	Serving GPRS Nodes
GGSNs	Gateway GPRS Support Nodes
NTMP	Network Traffic Mining Platform
DPI	Deep Packet Inspection
STD	Standard Deviation
DTW	Dynamic Time Warping
Soft-DTW	Soft Dynamic Time Warping
SBD	Shape Based Distance
NCCc	Cross Correlation with Coefficient Normalization

SIL	Silhouette index
DB	Davies-Bouldin index
DB*	Modified Davies-Bouldin index
D	Dunn index
COP	COP index
ARMA	Autoregressive Moving Average
MA	Moving Average
AR	Autoregressive
SARIMA	Seasonal Autoregressive Integrated Moving-Average
SVM	Support Vector Machines
SVC	Support Vector Machines Classification
RBF	Radial Basis Function
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Network
DeepAR	Deep AutoRegressive
GAM	Generalized Additive Models
AWS	Amazon Web Services
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

Over the last few years, the traffic load on wireless networks has been increasing exponentially. Ericsson Mobility Report stated that over the past two years (2018 - 2019), data traffic increased by 49%, reaching up to 40 Exabytes [1]. Although it is generally agreed that the increasing amount of data used by smartphone users is a major factor for this growth, the introduction of data usage in many smart devices is another reason of this rapid growth. This widespread growth has introduced numerous challenges. One of these challenges, addressed in this research, is the prediction of surges in data traffic and base station power consumption.

1.1 Problem Statement

One of the consequences of the enormous growth in data usage is the need for more base stations in the network to serve more devices. However, when more base stations are added, a significant increase in energy consumption occurs. Base stations require energy equal to approximately 70% of the total energy of the entire network infrastructure [2]. With the ability to predict the traffic on base stations, the opportunity to save energy arises by developing a method to switch off base stations during the off-load times [2]. This research is centered around the problem of network traffic load prediction on base stations, while focusing mainly on exploring the most dependable and accurate method that can be used for forecasting traffic load.

1.2 Thesis Objective

The objective of this research is to assess the most popular time series forecasting techniques to forecast the estimated traffic load on base stations within a permissible margin of error. This is outlined in the following four steps:

1. Selecting a public dataset that can be used for the task of forecasting the traffic load of base stations.
2. Analyzing the selected dataset for extracting useful information and applying the required data preprocessing techniques on it.
3. Clustering the base stations according to their behavior before applying the forecasting techniques and examining the effect of clustering on the accuracy of the forecasting results.
4. Comparing the performance of commonly used time series forecasting techniques; these are statistical, machine learning and deep learning methods and how they can be in forecasting the traffic load of base stations.

1.3 Thesis Contributions

The contributions of the research presented in this thesis can be summarized as follows:

1. Data Analysis and Preprocessing

   First, a thorough analysis was applied on the selected public dataset "City Cellular Traffic Map" [3]. The effects of all the existing features on the traffic load were explored by performing analysis of all features. Then, the fluctuations of the traffic load over the seven days were analyzed over every day and every hour, and when it was noted that the load reached the peak. Next, the stationarity of the traffic load series was analyzed. Finally, the limitations of the dataset were reviewed.

   Second, preprocessing steps were applied on the dataset. These steps can be divided into three stages: data cleaning, feature engineering, and data transformation. The data cleaning included removing base stations with only a few records, removing outliers, identifying missing data, and replacing undefined values. For the feature engineering, new features were derived by

applying different functions on the data. Finally, transformation took place to normalize the data and convert the nonstationary series to a stationary one.

2. Clustering base stations

   We examined the effect of clustering similar data before training the forecasting techniques. For the clustering process, different clustering algorithms with different distance metrics were explored. The performance of each algorithm was analyzed using difference error metrics in order to isolate the clustering algorithm with minimum clustering error. To the best of our knowledge, this is the first work that uses similarities based clustering of base statins according to their behavior.

3. Forecasting the traffic load of base stations

   Several techniques were applied to perform the forecasting of the traffic load of base stations including statistical methods, machine learning methods, and deep learning methods. Hyper parameter tuning was applied to improve the accuracy of these models. Two online tools created for time series forecasting were used with the experimented dataset and those results were compared to those of other experimented models. Lastly, the results of each method were presented, and compared with respect to the performance of each model as a solution to the forecasting problem.

1.4 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 reviews recent literature on the use of network traffic load prediction along with its importance and the most common techniques used to accomplish the predictions. Chapter 3 describes the dataset used in this research, how it is analyzed with discussion of its merits and limitations. In addition, the preprocessing work

performed on the dataset is presented. Then, Chapter 4 explains the need and importance of the clustering process for the project. After that, Chapter 5 presents the results of the forecasting algorithms. In addition, a comparison between their performances is discussed. Finally, Chapter 6 concludes the research findings, and suggests future work.

Chapter 2

Background and Literature Review

This chapter is divided into three main sections. The first section explains the most common algorithms used for time series clustering and the similarity metrics that can be used in these algorithms. The second section illustrates the popular time series forecasting methods, the statistical methods used in time series forecasting, the machine learning and deep learning algorithms that can be used in this research. Finally, the third section provides a literature review on network traffic prediction, and, in particular, base station load prediction. The first part of this section presents an overview of the traffic load prediction problem. The second part focuses on the base station load prediction and its importance. The last part reviews the recent research that examines different time series forecasting methods.

2.1 Clustering

Clustering is the process of grouping the data points into a number of groups where the similarity between that data points lying in the same cluster is higher than the similarity between them and the data points lying in the other groups. In this research, the focus is on clustering the base stations according to their behavior. The traffic load of the base station through the day can be represented as time series. Hence, we investigated different time series clustering techniques to accomplish the clustering task in this research. For clustering time series data, two main decisions should be made, the similarity measure or the distance metric that is used to measure the similarity between two time series and the clustering algorithm that is used to cluster the time series. The following two sections illustrates some of the most common time series similarity measures and clustering algorithms used in literature.

2.1.1 Time Series Similarity Measures

Many metrics exist to measure the similarity between two time series. In this research, the following metrics are considered.

1. Euclidean Distance

Euclidean distance is a famous clustering distance measure. Euclidean distance is calculated using the following formula [5]:

$$Distance_{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \qquad (2.1)$$

where: n is the space dimension, x is a vector that represents the coordinates of the first point and y is a vector that represents the coordinate of the second point.

The Euclidean distance can be used in calculating the similarity between two time series by calculating the square root of the sum of the squared difference between all the corresponding points lying on the two series. As shown in Figure 2.1, if *a* and *b* were two time series representing the behavior of two different base stations, then the Euclidean distance between them would be the square root of the sum of the squared length of the grey lines [6].

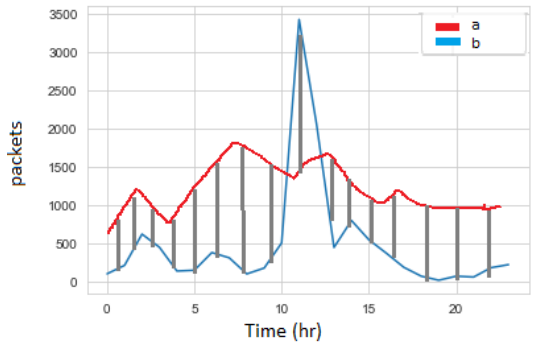


Figure 2.1: Euclidean distance between two time series. Redrawn from [6]

2. Dynamic Time Warping

DynamicTime Warping (DTW) is a dynamic programming technique that measures the distance between two series that can be equal or unequal in length and may also vary in time and/or speed based on the similarity between them [7].

DTW calculates the distance between two series by finding all of the paths between them and chooses the path with minimum distance using a distance matrix [8]. Each element in the distance matrix represents the cumulative distance of the minimum of the surrounding elements [8]. This can be explained as follows: Let us assume two time series *Q* and *C*, where *Q* is a time series containing *m* data points and *C* is a time series of *n* data points. A matrix of size *n* by *m* is generated where each element in the matrix represents the cumulative distance between the two opposite data points of the two time series and the minimum distance of the three surrounding elements. For instance, for a data point *i* belonging to the time series *Q* and a data point *j* belonging to the time series *j*, element *e* in the matrix is calculated by:

$$e_{ij} = d_{ij} + \min\{e_{(i-1)(j-1)}, e_{(i-1)j}, e_{i(j-1)}\} \quad (2.2)$$

where $dij = (ci + qj)^2$ [8].

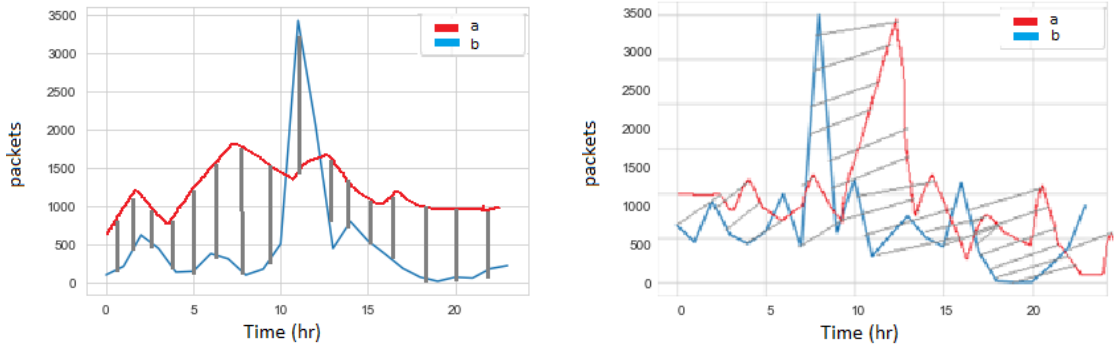
The best path is chosen using the path that results in the minimum distance at the last element in the matrix $e_{n,m}$. The DTW distance can be calculated by:

$$D_{DTW}(Q, C) = \min_{\forall\, w \in P} \left\{ \sqrt{\sum_{k=1}^{K} d_{\omega k}} \right. \quad (2.3)$$

where *P* is a set of all possible warping paths, and ωk is (*i*, *j*) at k_{th} element of a warping path and *K* is the length of the warping path [8].

It can be depicted that DTW is more flexible than Euclidean distance. For instance, if we have two time series, such as those in Figure 2.2, the two series have very similar behavior, but the second

series is shifted. If the Euclidean distance was used (Figure 2.2.a), the distance between the two series would be larger than if the DTW was used (Figure 2.2.b) [6].



(a) (b)

Figure 2.2: Euclidean distance vs DTW distance. Redrawn from [6]

3. Soft Dynamic Time Warping

Soft Dynamic Time Warping (Soft-DTW) is a differentiable loss function for time series. Soft-DTW uses the smoothed formulation of DTW and computes the soft-minimum of all alignment cost. The DTW has quadratic time but linear space complexity. The advantage of soft-DTW over DTW is that for both the value and gradient can be computed with quadratic time/space complexity [9].

4. Shape Based Distance

Shape Based Distance (SBD) is another distance measurement technique that was claimed to be faster than the traditional DTW [10]. The SBD is sensitive to scale because it depends on the cross correlation with coefficient normalization (NCCc) which is produced using the convolution between the time series for which the distance is calculated.

SBD is calculated using the following formula:

$$SBD\ (x, y) = 1 - \frac{\max(NCCc(x,y))}{\|x\|_2 \|y\|_2} \qquad (2.4)$$

where x and y are two vectors, NCCc is the cross correlation with coefficient normalization of x and y, and $\| \ . \|_2$ is the l_2 norm or the Euclidean norm of the series [10] [11].

The l_2 norm is calculated as follows:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.5}$$

and NCCc is calculated as follows:

$$NCCc(x,y) = \frac{\mathrm{CCw(x,y)}}{\mathrm{R_0(x,x)} \cdot \mathrm{R_0(y,y)}} \tag{2.6}$$

Where,

$$CC_w(x,y) = R_{w-m}(x,y) \tag{2.7}$$

$$R_k(x,y) = \sum_{1=1} x_{1+\kappa} yi \, , if \ k \geq 0 \tag{2.8}$$

$$R_k(x,y) = R_{-k}(y,x), if < 0 \tag{2.9}$$

where: m represents x and y size and w ∈ {1, 2, …. , 2m−1} [12].

2.1.2 Clustering Algorithms

After exploring the similarity metrics, the clustering problem turns out to be an optimization problem, achieving best solution from all available solutions, which has a plethora of analysis in literature. Many clustering algorithms exist in a broad range of areas to solve different problems. Clustering algorithms differ from each other in the way they deal with the data and how each cluster is created [13].

In this research, two algorithms are considered for clustering:

1. Partitional Clustering (K-means)

K-means is a common and simple clustering algorithm that can be easily applied to cluster groups in various problems. K-means algorithm is used to divide the data into K sets. K-means gives the

best results in case of compact and hyper spherical clusters [14]. The goal of K-means is creating clusters with highest possible similarity between the data inside each cluster and minimum similarity between the data in different clusters. The K-means objective function can be defined as:

$$J=\sum_{j=1}^{k}\sum_{i=1}^{n}\left\|x_i^{(j)}-c_j\right\|^2 \quad (2.10)$$

where: *J* represents the objective function, *K* represents the number of clusters, *n* represents the number of data points, $x_i^{(j)}$ represents data point *i*, c_j represents the centroid for cluster *j* and $\left\|x_i^{(j)}-c_j\right\|^2$ represents the distance function [15].

The K-means algorithm works by initially choosing k cluster centers and then assigning each data point to its nearest cluster center, then updating each cluster center by the mean value of the belonging data points until no change is detected [16]. The K-means algorithm is known to have a computational complexity of O(KNd) where K is the number of clusters, N is the number of data points and d is the number of dimensions [14]. The value of K can be chosen randomly or by trying different numbers and choose the number with best accuracy. Two of the most popular techniques that are used to determine for the K-means clustering evaluation are the Elbow method and the Silhouette score. The Elbow method uses the fact that when the number of clusters increases the distortion will decrease. Hence, it uses an initial value for k and calculates the distortion resulted from the clustering for each value from 0 to k. Then the number at which the distortion stops to decrease with the same rate as the previous rates is considered as the chosen number of clusters [17]. However, it was shown that the elbow method is usually not very accurate [18]. Hence, in this research the clustering was not used in determining the k value. Instead, the Silhouette score was used with other metrics and they are explained later in Section 4.7.

2. Hierarchical Clustering

Hierarchical Clustering is another popular clustering technique that is developed to group similar clusters in a form of a hierarchy or tree. Hierarchical Clustering can be divided into two types: Agglomerative clustering or bottom-up clustering and Divisive clustering or top-down clustering. The agglomerative clustering considers all of the existing data points as distinct clusters, then lumps two clusters that have the minimum distance and considers them a cluster until all of the points are combined in one hierarchical cluster [19]. The hierarchy that presents which clusters were formed in each iteration is called the dendrogram and it is used to provide the clusters of a required length when broken into links [20]. Divisive clustering differs from the agglomerative clustering in that it initially considers all of the data points as one cluster, then it keeps dividing them into more similar clusters until it reaches one cluster for each data point. Agglomerative clustering is less complex and more common than the divisive clustering. Hence, the focus in this research was about the agglomerative clustering.

The first step in agglomerative clustering is calculating the distances between each pair of data points in a proximity matrix. Second step is merging the pair with the minimum distance in one cluster and updating the proximity matrix with the new merged clusters and new distances. This step is repeated recursively until all the clusters are combined into one. The final step is dividing the tree at a predetermined height that results in the desirable number of clusters [21]. Agglomerative clustering can follow one of the following three methods to measure the distance between the clusters [22]:

1. Complete Linkage

   The Complete Linkage calculates the distances between all pairs of data points in the clusters. Then it considers the maximum one as the final distance.

$$\forall C_i, C_j L_{ij} = max\{\, d(x_a, x_b)\, \forall x_a \in C_i \; and \; x_b \in C_j \,\} \tag{2.11}$$

2. Average Linkage

The Average Linkage calculates the distance between the clusters as the average distance between all pairs of data points in the two clusters.

$$\forall C_i, C_j L_{ij} = \frac{1}{|C_i||C_j|} \sum\nolimits_{x_a \in c_i} \sum_{x\_b \in c\_j} d(x_a, x_b) \tag{2.12}$$

3. Ward's Linkage

The Ward's Linkage uses the aggregation of the squared distances between all pairs of datapoints in the two clusters.

$$\forall C_i, C_j L_{ij} = \sum\nolimits_{x_a \in c_i} \sum_{x\_b \in c\_j} \|x_a - x_b\|^2 \tag{2.13}$$

In the above three equations, C_i, C_j represents two different clusters, L_{ij} represent the linkage method and *d* represents the distance function used to calculate the distance between the datapoints in the cluster.

The agglomerative clustering algorithm complexity is $O(N^2Log(N))$, Hence, it requires a lot of time when clustering large datasets [22].

2.1.3 Time Series Analysis Tools

Some packages providing tools for the analysis of time series exist. These tools provide methods for the clustering problem. In this research, the following two packages were used and compared:

1. Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance (dtwclust)

dtwclust is an R Package created for Time Series Analysis generally and Time Series Clustering specifically. dtwclust is an R library that was developed for the analysis, clustering and visualization of time series data [13]. It provides an implementation of the traditional clustering

techniques, such as partitional and hierarchical clustering, that can be used for the time series data [13]. The dtwclust package offers an option to provide the time-series centroid or time series protocol in other calls. Many options can be taken for choosing the centroid, but in this research, the following two were examined:

1. Mean

The centroid here is calculated by taking the average of the summation of each point i in all of the time series in the cluster [10]. The mean can be calculated by the following formula:

$$u_i^p = \frac{1}{N} \sum_t x_{c,i}^p \, , \forall \, t \in C \qquad (2.14)$$

where: C represents the cluster, N the size of the cluster, and $x_{c,i}^{v}$ represents the element i of the point p from the time series t that belongs to cluster C [10].

2. Partition around medoids

Another approach is the partition around medoids, which differs from the Mean method in that the centroid is a representative object from the cluster itself [10]. It was claimed that sometimes partition around medoids provides better results than the mean method because if the medoid was a member of the cluster then the time series structure would not be altered [10]. In the dtwclust library, partition around medoids is implemented as follows:

First the rudimentary centroids are chosen randomly. Then the distance between all the data points in the time series and these centroids is calculated. After that each data point is connected with the closest centroid to it. Next, for each cluster, the distances between all the time series belonging to that cluster are calculated. Finally, the time series that results in the least distance is selected as the new centroid. This procedure keeps iterating until no further change is noticed [10].

2. tslearn

tslearn is a Python package created for the analysis of time series data, built on scikit-learn, numpy and scipy libraries [23]. For the clustering process, tslearn provides three methods:

1. GlobalAlignmentKernelKMeans

Global Alignment Kernel K-means applies the kernel k-means algorithm, which is an incremental algorithm that adds one cluster on each step through a global search method that depends on kernel K-means. Incremental K-means works as follows: First, the traditional K-means is run to define the initial clusters. Second, threshold value is defined as the maximum allowed dissimilarity between datapoints in one cluster. Third, the centroid of any cluster is chosen as a source point and another point is selected randomly from the dataset. Fourth, the distance between the two datapoints is computed. Fifth, if the distance is smaller than the defined threshold, the selected datapoint is merged with the selected centroid cluster the cluster centroid is updated. If the distance is larger than the defined threshold, a new cluster is created, and the selected point is assigned as its centroid. The last three steps are repeated until all the data points are clustered [24] [25]. This incremental procedure can reduce the probability of reaching local minima solutions [26].

2. KShape

KShape is a type of partitional clustering algorithms that keeps the time series shapes. KShape algorithm uses the SBD distance measurement to compute the centroids of each cluster [23]. Using the shape property, it keeps updating the clusters centers and pair the centroids with the data points with shortest SBD until no further change occurs.

3. TimeSeriesKMeans

TimeSeriesKMeans is the third existing algorithm in the tsclust library. It applies the famous K-means clustering algorithm for time-series data.

2.2 Forecasting

In recent literature many techniques for the time series forecasting problem are researched. These techniques vary in many aspects such as their dependencies, implementation and complexity. Several techniques perform the forecasting process by employing basic statistical methods and applying Artificial Intelligence concepts. The techniques vary in complexity, accuracy and performance. In this research, several techniques including statistical methods, machine learning, deep learning, and online tools were explored for performing the base stations load forecasting.

2.2.1 Statistical Methods

In this research, the following techniques are explored.

1. Autoregressive Integrated Moving Average (ARIMA).
2. Seasonal Autoregressive Integrated Moving-Average (SARIMA).

1. **Autoregressive Integrated Moving Average (ARIMA)**

The Autoregressive Moving Average **(**ARMA) model is one of the most commonly used techniques used in time series modeling and time series forecasting. ARMA model generates the future step in a series as a linear function of the records and errors from previous steps.

ARMA model integrates both the Autoregressive (AR) model, which predicts future values using linear regression of the current values of the series against previous values [27], and the Moving Average (MA) model, which uses the past errors multiplied by a coefficient for future predictions. In the ARMA model, p and q parameters are used for defining the AR order and the MA order respectively. The Autoregressive Integrated Moving Average (ARIMA) model is a higher

generalization of the ARMA model. ARIMA and ARMA can be tantamount in the combination of AR ad MA models, but the ARIMA model contains a component that the ARMA model does not have, which is the Integrative part (I). By this addition, Integration can be used to convert the non-stationary time series into stationary ones. Hence, the ARIMA model can occasionally be used with non-stationary data [28].

Like the ARMA model, the ARIMA model defines the AR order and the MA order using the p and q parameters, but ARIMA also uses a d parameter to define the order of the Integrative part. The forecasting equation of the ARIMA model can be presented as follows:

$$\hat{y}_t = \mu + \varphi_1 y_{t-1} + .. + \varphi_t y_{t-p} - \emptyset_1 e_{t-1} - .. - \emptyset_q e_{t-q} \quad (2.15)$$

where: μ is a constant, y is the output at time t, $\hat{y}_t$ is the predicted output, φ is the coefficient of each p parameter, $\emptyset$ is the coefficient of the q parameter, $\varphi_1 y_{t-1} + .. + \varphi_t y_{t-p}$ represent the AR terms and $-\emptyset_1 e_{t-1} - .. - \emptyset_q e_{t-q}$ represent the MA terms.

2. Seasonal Autoregressive Integrated Moving-Average (SARIMA)

Seasonal Autoregressive Integrated Moving-Average (SARIMA) is an extension of the ARIMA model that handles the seasonality that may exist in the data. Seasonality in data indicates that there exists a pattern that keeps repeating every season or every number of time steps [29].

The SARIMA model notations can be defined as follows: ARIMA (p, d, q) x (P, D, Q) S; : where p represents the non-seasonal AR order, d represents the non-seasonal differencing, q represents the non-seasonal MA order, P represents the seasonal AR order, D represents the seasonal differencing, Q represents the seasonal MA order, and S represents the time span of repeating seasonal pattern. The general equation of the SARIMA model can be formulated as follows [30]:

$$\Phi(L^s)\varphi(L)\Delta^d \Delta_s^D y_t = \theta_0 + \Theta(L^s)\theta(L)_t \quad (2.16)$$

where Δ^d specify the differencing order and Δ_s^D specify the seasonal differencing order.

And:

$$\varphi(L) = 1 - \varphi_1 L - \ldots - \varphi_p L^p \quad (2.17)$$

$$\theta(L) = 1 - \theta_1 L - \ldots - \theta_p L^p \quad (2.18)$$

$$\Phi(L^s) = 1 - \Phi_1 L^s - \ldots - \Phi_p L^{Ps} \quad (2.19)$$

$$\Theta(L^s) = 1 - \Theta_1 L^s - \ldots - \Theta_p L^{Qs} \quad (2.20)$$

where: φ and θ are the lag operators and Φ and Θ are the seasonal operators.

2.2.2 Machine Learning and Deep Learning

2.2.2.1 Machine Learning

In this research, several machine learning techniques are used. Since in our problem the target value for forecasting is the load value, which is a continuous value not a discrete one, the problem was considered a regression problem. Hence, the following regression algorithms are examined:

a) Support Vector Regression

b) Multi-layer Perceptron Regression

c) Decision Trees Regression

d) Random Forest Regression

e) Extreme Gradient Boosting (XGBoost)

a) Support Vector Regression

Support Vector Regression (SVR) is considered a special type of Support Vector Machines (SVM), which is a supervised learning algorithm that aims to classify the data points by finding the hyperplane that can separate the data with minimal error. For instance, in Figure 2.3, the SVM would aim to find the red line as a hyperplane to separate the green circles and blue squares.

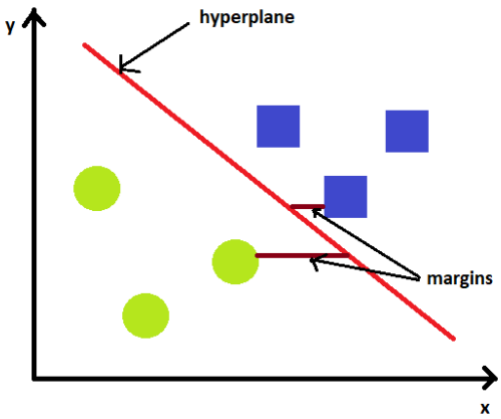


Figure 2.3: SVM Hyperplane.

The hyperplane can be considered as a boundary that splits the data into multiple groups. In case of two-dimensional data, the hyperplane can be represented as a line. In case of three-dimensional data, the hyperplane can be a plane. For higher dimensions it is called a hyperplane. The best hyperplane is the hyperplane with the maximum margin. The margin can be defined as the distance between the hyperplane and the nearest point from each class.

In case of nonlinearly-separable data, where n-dimensional data cannot be separated using n-1 hyperplane, SVC usually converts the data into higher dimensions using a technique called Kernel Trick. The kernel trick idea is to keep converting the data using a kernel function into higher dimensions until a dimension is found where the data can be separated using a hyperplane. The kernel function is usually either linear, polynomial, or radial. The kernel is considered as a dot product in an n-dimensional feature space and the most common kernel functions are linear kernel, polynomial kernel and Radial Basis Function Kernel. For instance, for two vectors x and y the kernel functions formulas would be as follows:

- Linear Kernel: $k(x, y) = x^T y$
- Polynomial Kernel: $k(x, y) = (1 + x^T y)^2$
- RBF Kernel: $k(x, y) = \exp(-\gamma \| x - y \|^2)$

where: γ: gamma and calculated from $\frac{1}{2\varphi}$ [31] where φ: phi is a free parameter. Gamma is used to control overfitting when it increases the tendency of the model to over fit increase and vice versa. The SVM algorithm has many positive features. First, it is very powerful in case the classes are separable [32]. Also, it shows efficiency in case of high dimensions data. Moreover, it can handle outliers very well. However, SVM also has a few shortcomings. SVM is not ideal in large datasets because it takes high processing time. It is also not efficient enough when classifying overlapped classes [32].

Support Vector Regression (SVR) applies the same concepts as the SVM, but since the output of the regression is a continuous value, in SVR there is a margin of tolerance (epsilon) that the error should fit in. For instance, in Figure 2.4, the goal of the SVR algorithm would be to find a line y=wx+b that has maximum distance ε from the original data points [15] [33].

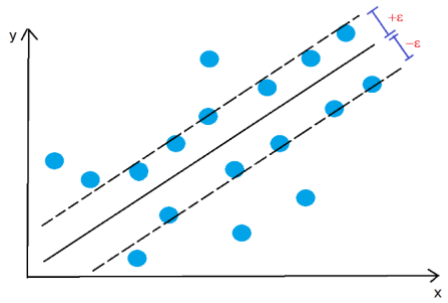


Figure 2.4: SVR model. Redrawn from [15]

b) Multi-layer Perceptron Regression

Multi-layer Perceptron (MLP) is a basic type of Artificial Neural Networks. It was developed to solve the non-linearity problem that could not be solved using a simple Perceptron [34].

The perceptron is considered as the base of the neural networks. As shown in Figure 2.5, the perceptron consists of a few inputs, weights, bias, output and an activation function.

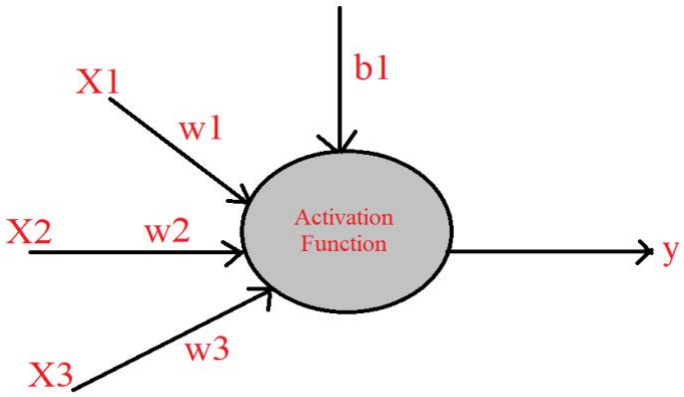


Figure 2.5: Perceptron Architecture

A perceptron may include more than one input, but it only results in one output. The perceptron calculates the output by applying the activation function to the sum of the products of the weight with the inputs plus the bias. The perceptron formula can be represented as follows [35]:

$$y = f(\sum_{i=0}^{n} wi * Xi + b) \quad (2.21)$$

where: y is the output, w is the weights vector, x is the input vector, b is the bias, and f is the activation function.

The perceptron learns the correct behavior by calculating the difference between the correct output and the predicted output. Then using this error and a predefined learning rate, it updates the weights based on them. This step is repeated until no error is produced [33].

There are several activation functions that can be used in the perceptron. They can be divided into linear activation functions and non-linear activation functions. Identity function is an example for the linear activation functions. In addition, sigmoid, tanh, relu and leaky relu functions are examples for the nonlinear activation functions. These functions can be calculated as follows:

- Identity function: f(x)= x
- Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh function: f(x) = tanh(x)

- Relu function: f(x) = max (0, x)
- Leaky Relu function: f(x) = max (0.1x, x)

Perceptron were originally created for the classification task because the output is usually either 0 or 1 or either -1 or 1 depending on the used activation function. However, perceptron can be also be used for regression if the activation function is discarded.

Although perceptron can solve various types of problems, it failed to solve any non-linearly separable problem [36]. The interest in perceptron decreased because of this problem, when it failed to solve a simple XOR logical function. The interest towards the perceptron and neural networks rose again in the 1980s when they combined many layers of perceptron to form the Multilayer Perceptron (MLP) and were able to use it as a solution for the nonlinearly separable problems. MLP is a supervised learning algorithm that performs feed-forward networks. MLP networks consist of an input layer that consists of a set of features $X=x_1, x_2, \ldots\ldots, x_m$, then an n number of hidden layers and an output layer. A simple representation of the MLP is shown in Figure 2.6.

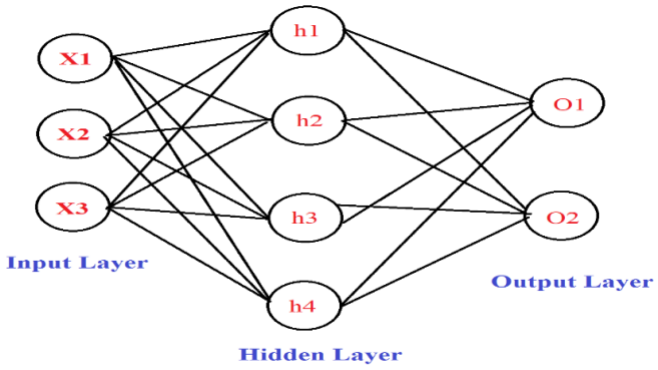


Figure 2.6: MLP Architecture

MLPs works on approximating a function f, that $Y^n = f(.) * X^m$ where m is the number of input features, n is the number of outputs and * means the application of the function on the input vector.

For the hidden layer, each one is represented by h = f(Wx + b) where f is the activation function W is the set of weights connected to the input and x is the input of the previous layer and b is the bias. The output is calculated by calculating the result of an activation function on the sum of the product of the weights with the output of the hidden layers. This step where the signals flow from the input to the output passing by the hidden layers is called forward propagation. After the forward propagation step, the error is calculated using a loss function. Several loss functions can be used. The loss function choice depends on the goal of the network. If the network was designed for a classification task, then the cross entropy (eq. 2.22) is commonly used. If the goal was regression, then the mean squared error (eq. 2.23) or the mean absolute error (eq. 2.24) loss functions can be utilized [37] [38] [39].

Cross-entropy: $$\text{Loss} = -y \cdot \log(\hat{y}) \quad (2.22)$$

Mean Squared Error (MSE): $$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}(yi - \hat{y})^2 \quad (2.23)$$

Mean Absolute Error (MAE): $$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}|yi - \hat{y}| \quad (2.24)$$

where: $\hat{y}$ represents the correct output and yi represents the predicted output.

To decrease the total loss function, diffrentiation of each function in the network starting from the error function moving to the output layer activation function and the hidden layers activation functions takes place. The importance of the diffrentiation step is that it allows the network to demonstrate the effect of any change in the error on the all of the internal weights. Also, the decomposition of the derivatives allows for backpropagation. After diffrentiation of the functions in the network, the error is backpropagated from the output layer to the hidden layers then to the input layer. Finally to update the weights, the new value of the weights resulting from the backpropagation step is multiplied by a predetermined learning rate and then get subtracted from the old weight. This step is calculated using the delta rule which is presented as follows:

$$w_{new} = w_{old} \propto - \frac{\partial \mathrm{E}}{\partial \mathrm{w}} \quad (2.25)$$

where: w represents the weight, $\propto$ represents the learning rate and $\frac{\partial \mathrm{E}}{\partial \mathrm{w}}$ represents the partial derivative of the error on the weight.

Then the processes of forward and back propagations are repeated to update the weights until the error is minimized or a determined number of epochs [34].

The difference between regression and classification for the MLP is that in regression, the backpropagation is performed with no activation function nor an identity function in the output layer [40] [33].

c) Decision Trees Regression (DecisionTreeRegressor)

Decision Trees are another supervised learning algorithm that can handle linear and nonlinear problems. Decision Trees work by creating a tree-like model of decision questions. Figure 2.7 illustrates a basic structure for the decision trees. Where each node represents a question that it will split into another node or provide a result in a leaf. Features are treated as questions in the tree. The prediction of each record is obtained by following the path resulting by answering the decision questions according to the record values [41].

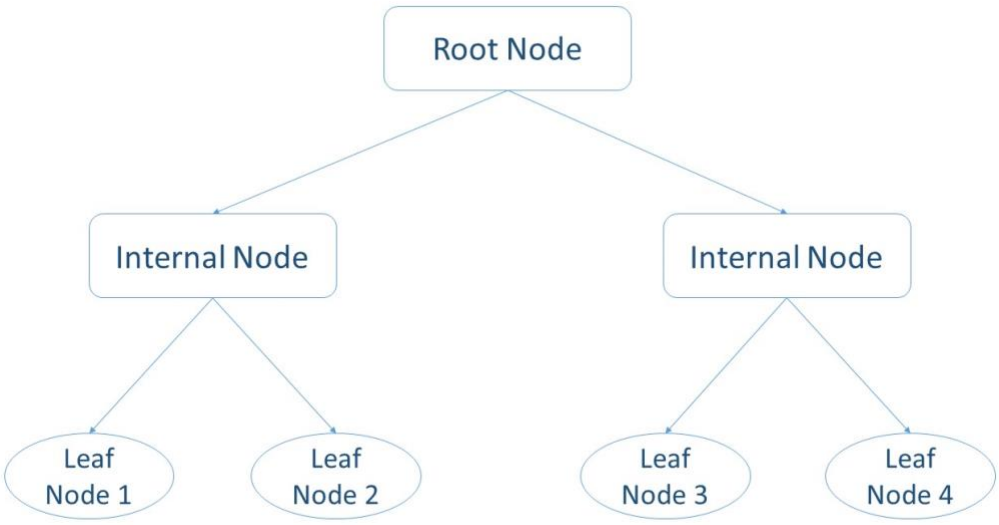


Figure 2.7: Decision Trees Basic Structure

The Decision trees can be grouped into two classes according to the type of target variable: Categorical Variable Decision Trees and Continuous Variable Decision Trees. In Categorical Variable Trees, the target is categorical which is something that can be answered using true or false, for instance, a feature that determines if a person is infected or not. While in Continuous Variable Trees, the target has a continuous value such as numeric values or date and time.

The Decision trees can be built using different algorithms. The most famous algorithm used for building the trees is ID3. ID3 is a top-down greedy search algorithm that works as follows:

- Select a random feature f from the data as the root node
- Create an internal node based on each value of f
- Using a criterion function, calculate the performance of the tree
- Repeat the previous steps for each feature in the data
- Select the feature with the best performance and split the tree using it
- Repeat the previous steps for each internal node with the rest of the features until the training data are classified correctly using the tree

Several functions can be used as the criterion method in the decision trees. Most famous techniques are Gini Impurity (eq. 2.26) and the Information Gain (eq. 2.28) which can be calculated as follows [42]:

$$\text{Gini}=1-\sum_{i=0}^{n} pi^2 \tag{2.26}$$

$$\text{Entropy}=\sum_{i=0}^{n} -pi * \log(pi) \tag{2.27}$$

$$\text{Information Gain} = Entropy_{root\ node} - Entropy_{child\ nodes} \tag{2.28}$$

where: n represents the number of classes which the data can be classified into and p represents the probability of each class

Decision Trees are mainly used for classification, but they can also handle regression problems. In classification the decision questions in the tree require a true or false answer to split the node while in regression Mean Squared Error (MSE) is usually used. The Decision Trees Regressor searches all the distinct values of all features and splits the node on the feature that provides the minimum error metric. The final output in classification trees is binary but it is a continuous value in case of the regression [33].

d) Random Forest Regression (RandomForestRegressor)

Decision Trees usually perform well when the output is included in the training data, but if the output value is different than the range of the values in the training data, the decision trees performance degrades [43]. Random Forests were developed to overcome this limitation and provide more flexibility. As shown in Figure 2.8, random forests are mainly built from several Decision Trees combined.

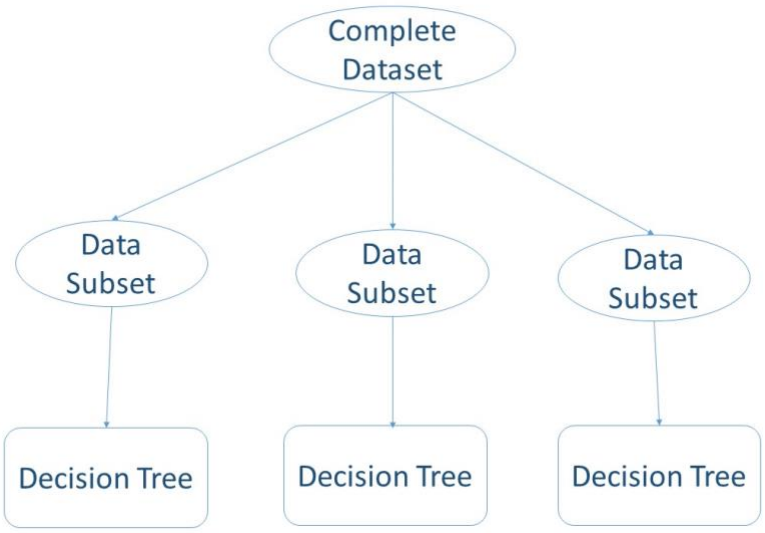


Figure 2.8: Random Forests basic Structure

The Random Forests algorithm is based on the bagging concept. Bagging or bootstrap aggregating is a technique developed for reducing variance without altering the bias. Bagging is considered when solving low bias high variance problems. It is based on combining weak learners by an aggregation technique, where each learner works on a different subset from the complete data. Most of the time, the outputs of weak learners are aggregated using their average value in case of regression problems or by the majority of the outputs in classification problems [44]. The weak learners can be parallelized because they are trained independently, and that can reduce the processing time required for the bagging task [44]. Random Forest adds to traditional bagging the idea of feature selection working as follows: First it creates a random subset from the existing dataset then it considers only a random subset of features at each step. Then it keeps repeating these steps for n number of times, this will result in n number of different trees. Finally, the training data will be run on all trees and each tree output will be considered as a vote for a prediction value. The final prediction will be the prediction that received the maximum number of votes or the average of the votes. By adding the feature selection in Random Forests, the models will be more

flexible and fast in case of missing data. This can be achieved because the trees that were trained on a subset of the features can classify the records that are passed to the model with other trees missed features values [44].

As in other algorithms, if the prediction is a binary value then the Random Forests classifier is used. Otherwise, the Random Forests regressor is chosen [33] [43].

e) Extreme Gradient Boosting (XGBoost)

XGBoost is a new powerful algorithm developed by Tianqi Chen and provided as an open source library [45]. XGBoost algorithm popularity has increased in the last decade among data scientists and competitors in machine learning challenges like Kaggle's. XGBoost term refers to "Extreme Gradient Boosting" because it is based on decision trees and gradient boosting [46]. Boosting is a concept tantamount to bagging in using weak learners to improve predictions. But unlike bagging where the learners are trained independently from each other and can be parallelized, the learners in boosting depend on each other and can only be trained sequentially. The most famous boosting algorithm is AdaBoost which stands for adaptive boosting [47]. AdaBoost works as follows: It starts by selecting a random subset of the training data and trains it in the usual way using a weak learner. Next it uses all the training data to test the weak learner model to discover the points which were not predicted correctly. Then, it selects another random subset from the training data, but each point is weighted according to the error resulted in the previous step. Hence, the points which were not correctly predicted become more likely to be chosen than the correctly predicted ones. Again, the training data is tested, but this time it will be tested using the old and the new learners. The results of the learners will be aggregated, and the error will be measured across all the data. These steps are repeated until a predefined number of weak learners is reached [33]. The difference between the gradient boosting and the adaptive boosting is that the misclassified points are not

weighted. The gradient boosting calculates the loss of the previous weak learner and tries to add a new weak learner that reduces the loss resulted from the previous weak learner. XGBoost inherits the gradient boosting idea and boosts the speed and the performance of the traditional gradient boosting. The gradient boosting is a sequential technique that requires a long time to process. XGBoost enhances the performance of the gradient boosting by some techniques such as introducing parallelization solution and applying distributed computing models for large datasets. The XGBoost algorithm works as follows: First, an initial tree is formed, and the similarity scores and the gain are calculated for it. Based on these values, it is determined how the data are split. Then the tree is pruned by calculating the differences between the Gain values and a hyper parameter that represents the Tree Complexity Parameter (gamma); if the difference is negative then the tree is pruned. Moreover, a regularization parameter lambda is used in calculating the similarity score to prevent over-fitting. The previous steps keep repeating until no improvement can be reached . In the XGBoost library, the booster can be set to: gbtree, dart or gblinear. The gbtree booster uses regression trees as the week learners. The dart adds to the gbtree a dropout technique to drop some learners randomly using the dropout rate in order to prevent overfitting. The gblinear differs from the dart and the gbtree in using linear regression instead of regression trees [46].

To sum up, XGBoost is considered a powerful boosting algorithm that usually outperforms normal other ensemble techniques because when in building new trees, the trees tend to perform better than the previous built trees and avoid their mistakes.

2.2.2.2 Deep Learning

In this research, deep learning, specifically Recurrent Neural Networks (RNNs), was used to examine its performance in predicting the network load on the base stations. RNNs are an advanced

type of neural networks. While traditional neural networks deal with the inputs and outputs of the network as individual components in the network, RNNs assume that future predictions can depend on past ones to allow previous outputs to be used as inputs using hidden states. Hence, RNNs can be defined as neural networks that can capture dependence over time (Temporal Dependence) [33].

Because RNNs make use of the past patterns, and since time series forecasting depends on the idea that the past behavior and patterns can be used to predict future values, RNNs can definitely be applied in time series forecasting.

RNNs apply the same concepts as those of the Artificial Neural Networks (ANNs), however RNNs differ in two major aspects from the ANNs. First, the input-output representation, at each time step single input single output is used for network training in ANNs, while in RNNs, since the previous output matters, the training is done with sequences. The second difference is the use of memory elements in the RNNs which represents the output of the hidden layer neuron and it will be counted as another input during next training step [48].

Folded model of RNNs can be illustrated in Figure 2.9, where Xt represents the input vector, Yt represents the output vector and S represents the state layer, Wx represents the weight matrix connecting the input to the state layer, Wy represents the weight matrix connecting the state to the output layer, and Ws represents the weights connecting the state from the previous time step to the state in the next time step. S is calculated by an activation function that requires the sum of the product of the input with the corresponding weight matrix and the product of the previous activation values with their corresponding weight matrix. Finally, the output is calculated by using an activation function on the linear combination for each input to each output node [48].

Figure 2. 9: Folded RNN visualization. Redrawn from [48]

Training in RNNs is like ANNs except that backpropagation should be done through time [48]. In Back Propagation Through Time, the network is trained at time step T but also considers all the previous time steps. That means that if W at time t=3 needs to be updated, the error would depend on W at time t=1, t=2 and t=3 [48].

Traditional RNNs achieve satisfying results as long as the time dependency is short, usually that means that the range of the considered previous time steps is between one and ten steps. If the number of previous time steps increased the RNN faces the vanishing gradient problem. The vanishing gradient occurs in the backpropagation process of the RNNs, when the weights matrices are adjusted with the use of a gradient. During the calculation of gradients by continuous multiplications of derivatives, the value of these derivatives may keep decreasing until the gradient is nearly vanished [48].

Many solutions were suggested to solve the short memory problem in the RNNs. In this research two of the most popular solutions that were used to solve the short memory problem were examined: LSTMs and GRUs. The novelty in these solutions is that they use gates to learn the past sequences that should be kept and the ones that should be forgotten. By doing so, it can pass the important information and track long-term dependencies [49] [48].

While both the LSTMs and the GRUs have the same goal of solving the vanishing gradient problem, they differ in the process they do to reach that goal. The LSTM architecture consists of four gates: forget, learn, remember, and use gates. The input gate determines what to be kept of

the new cell state, the forget gate determines what to forget from the existing memory, and the output gate decides what the next hidden state should be. The GRU architecture consists of two gates, a reset gate and an update gate. As their names imply, the reset gate determines the past information to be reset or forgot, and the update gate determines what should be used to update the new cell [48].

LSTM has the same architecture as RNN, except that instead of the normal neurons in RNNs, LSTMs have special kinds of cells. Figure 2.10 and Figure 2.11 show the LSTM cell and its architecture. LSTM architecture includes four different gates: forget gate, learn gate, remember gate, and use gate. In addition, LSTM uses two different types of memories; long- term memory and short-term memory. The long-term memory uses the forget gate where it forgets everything that does not seem useful. The short-term memory is combined with the current input to use the learn gate which learns recent information and neglects those that are unnecessary.Then the output of the forget gate which has not been forgotten yet is passed to the remember gate joined with the new information that has been learnt through the learn gate. The output of the remember gate defines the new long-term memory. Finally, the use gate determines the information that will be used from the previous long-term memory combined with the information that was just learnt in the learn gate. The output from the use gate determines both the current prediction and the new short-term memory [50] [48].

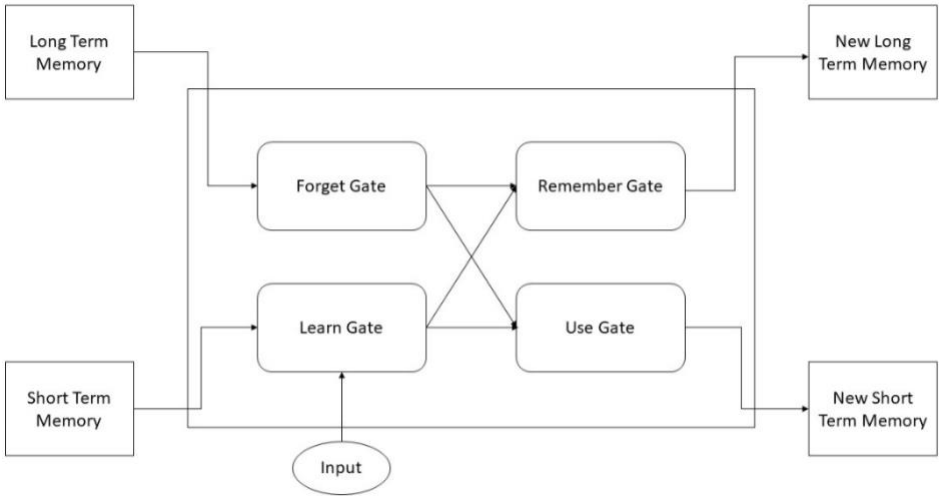


Figure 2.10: LSTM cell Architecture. Redrawn from [50].

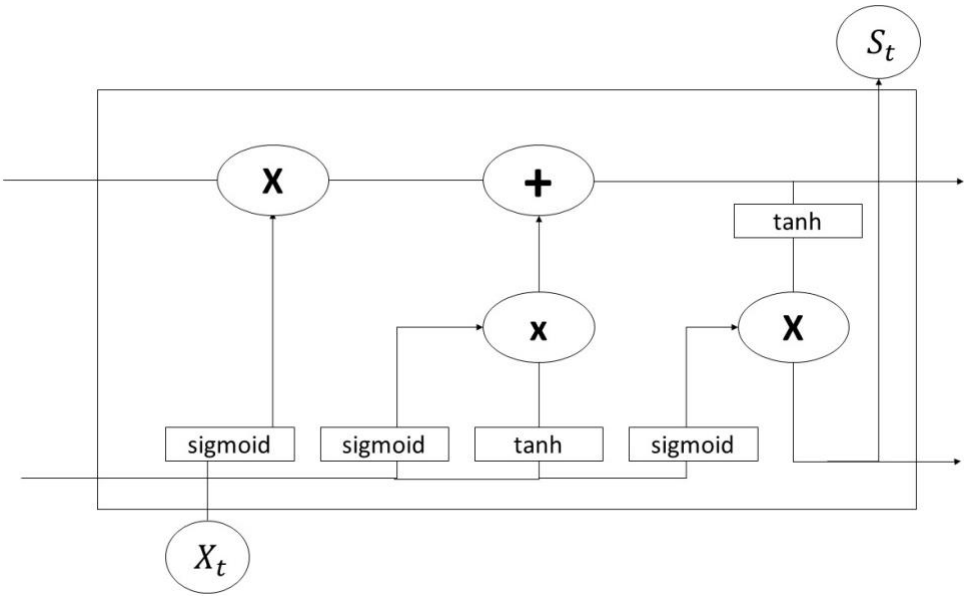


Figure 2.11: LSTM Cell. Redrawn from [50].

On the other hand, in GRUs, the forget gate and the learn gates are combined into Update gate. Also, instead of long-term memory and short-term memory, GRUs deal with one working memory. Figure 2.12 and Figure 2.13 show the GRU cell and its architecture. The update gate controls the amount of previous information that will be kept in the memory. If the update gate is around zero, then the new input is passed but no previous information is considered. The reset gate

determines the amount of previous information that will be dismissed from the memory. If the reset is near to zero, then the previous states will be dismissed [48] [51].

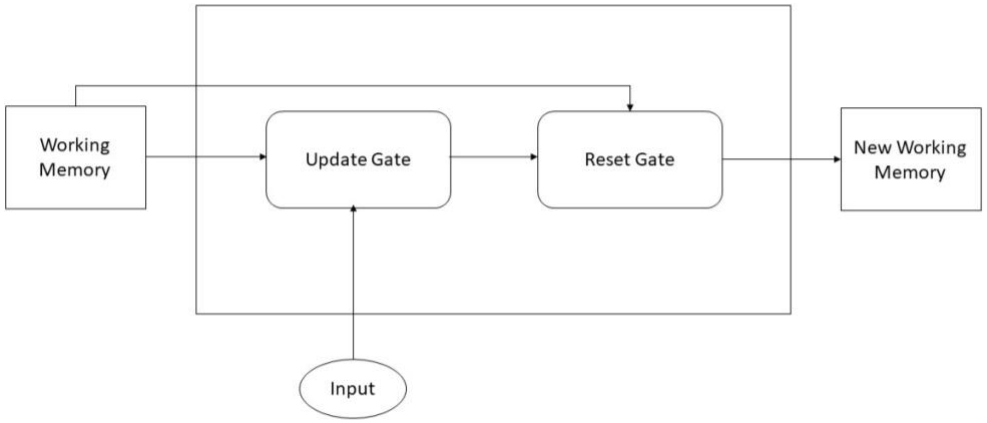


Figure 2.12: GRU cell Architecture. Redrawn from [48].

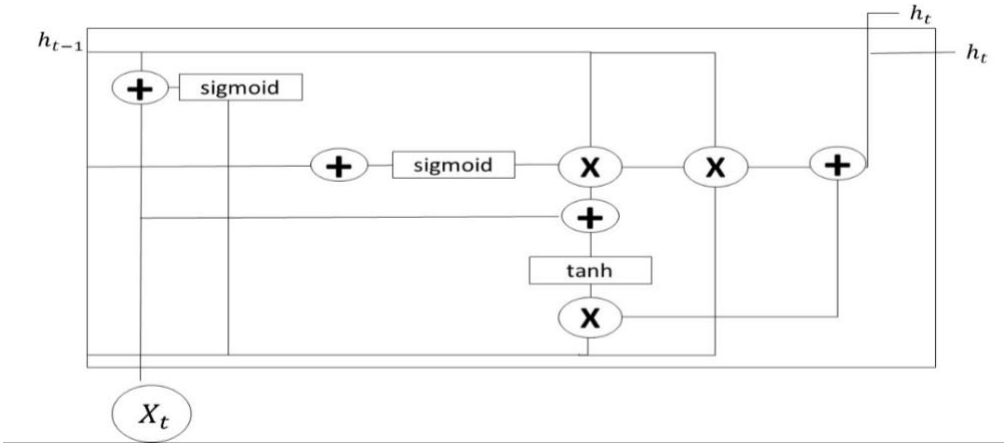


Figure 2.13: GRU cell. Redrawn from [48].

2.3 Literature Review

2.3.1 Traffic Load Prediction

Research on network traffic prediction varies between predicting the whole network load, predicting each user's load, predicting the number of users, or predicting the location of the user's smart devise that consumes network energy [52]. In the literature, one purpose of network traffic

load prediction is network planning [53]. Network planning is necessary to make the best use of the available resources and provide users with the best quality of service.

In [53], a network planning framework was developed in response to the network optimization problem. This framework introduced new services to users according to their demand. The prediction of each user's load was implemented to expect the demand on each service and the network was planned accordingly. The behavior of each user was observed from the vantage point of their data traffic load. The users were then grouped according to their behavior, and the services targeted each group independently.

Traffic prediction was also applied in Software Defined Networking-based IoT (SDN-IoT). In [54], a network traffic load prediction algorithm was developed to predict the fluctuations of traffic load. These predictions were used in the process of channel allocation in SDN-IoT. The authors mentioned that overlapping channels were considered during the channel assignment to the links to ensure that no congestion occurs. The authors claimed that their simulation results showed that using the traffic load prediction for the channel assignment process provided better results than other existing techniques.

A field where network traffic prediction was found to be useful is in wireless ad hoc networks (WANET) or Mobile Ad-hoc Network (MANET). Network traffic prediction can be used in enhancing the routing process which is considered a fundamental property of MANETs. MANETs are a wireless network that are built without existing infrastructure. Each node inside the MANET is considered an individual router. The routes between these nodes are usually multi- hop and dynamically constructed [55].

In [56], the issue of additional overhead being generated during the routing process was addressed. Several schemas were proposed that use network traffic prediction to reduce the time needed for route reconstruction, a mandatory step when the primary route between nodes breaks. The prediction results were used to solve the route breaks before they occur and perform the route reconstruction prior to the break. Simulation results showed that using network traffic prediction in MANETs was very effective in this area.

2.3.2 Base station load prediction

Predicting the load on base stations can be helpful in solving many wireless network problems such as energy saving. Since base stations consume more that 70% of the total cellular network's energy consumption [57], reducing the base stations energy consumption would consequently reduce the total network consumption.

In response to this issue, the authors in [58], made use of the base stations load prediction to create a base station sleeping mechanism. The output of the prediction was used to determine if the load would be low. If the base station was going to encounter low load, the base station would switch off and the base station's users would be switched to Pico Base Stations (PBSs) from Macro Base Stations (MBSs) [58]. When the prediction shows a foreseeable high load, the PBSs would switch off and the service would return to the MBSs. Simulation results showed that the PBSs need less power than the MBSs and that this methodology reduced the energy consumption in the network.

The authors in [59] utilized traffic load on base stations. They proposed a planning tool that uses the output of the base station load prediction along with other features to identify which base stations in the network should be turned on or off. The base station would be turned off if the load decreased and turned on if the load increased.

An additional energy saving schema was proposed in [60]. The future load prediction on the base stations was used to develop a grid-based energy saving scheme. Authors claimed that the base stations with low loads can switched into sleeping mode without any quality of experience degradation. In this schema, the authors take advantage of the overlapping coverage areas between neighboring base stations. Most often, networks are designed with extra base stations. Some areas may be served by several base stations where in fact the area can be served by fewer base stations. Hence, overlapping occurs. The schema considers the future prediction and checks if any base stations are overlapping. Then the extra base stations will be switched off until the future prediction shows they are needed, which in that case they will be switched back on.

Furthermore, predicting the traffic load on the base stations can be beneficial in defining the optimal placements of Aerial Base Stations [61]. Aerial Base Stations positions are dynamic. By predicting the traffic load on them, the predictions can be used to reallocate the base stations that are located in areas with no or low traffic load to areas that encounter high traffic load.

2.3.3 Prediction Techniques

Load forecasting processes have been applied in several ways, the most common method is using statistical methods such as Simple Moving Average (SMA), Exponential Smoothing, and ARIMA. Recently, research has moved towards Machine Learning, and Deep Learning methods, and most notably, RNNs to perform the task of load forecasting. There is also work that implements forecasting by using a comprehensive sensing method [62].

In [63], the authors compared the performance of the ARIMA model and the exponential smoothing model for predicting the load on base stations. The two models were examined using two scenarios. In the first scenario, the entire area was divided into multiple regions and the loads on all the base stations were predicted in each region together. In the second scenario, the

prediction of the loads on each base station were examined individually. From the two scenarios, it was found that the ARIMA model provided lower errors in predicting the load during the weekdays. But the exponential smoothing model was better at predicting the load on the weekends and was better for predicting the load on single cells [63].

Machine learning was used in [59] for performing the prediction process. The work in [59] used the past real records to create a SVR algorithm that performs the prediction. To build the SVR prediction model, the authors used the following features to construct the feature vector: chronological number of the time interval, the order of the day in the week, the number of the week of the training day, and the number of the year of the training day. Their target was the load on the base station. They ran the SVR algorithm using these values and divided their dataset to training and testing. Then, they assessed the performance on the testing data. Their training was divided to two phases. First, they used three weeks for training and the target was to predict the load of the next day for each day. They used eleven weeks for training and tried to predict the load for the next week. The authors claimed that they have achieved excellent results even with the change of load behavior [59].

In [4], an algorithm, based on Deep Learning for the load prediction called Hybrid Spatiotemporal Network (HSTNet), was proposed. HSTNet is a modification of the existing DenseNet that adds a deformable convolution layer [64]. Authors showed that HSTNet successfully captured the spatiotemporal characteristics in the traffic load and produced better prediction accuracy than the existing traditional statistical and machine learning techniques results [4].

Chapter 3

City Cellular Traffic Map Dataset Analysis and Preparation

In this chapter, a thorough explanation for the selected dataset "City Cellular Traffic Map Dataset" is presented. Then, the data analysis along with the data limitations are discussed. Lastly the preparation and pre-processing steps required on the data are demonstrated through discussing the data cleaning, data transformation and feature engineering techniques performed on the data.

3.1 Dataset Description

The dataset used in this project is the City Cellular Traffic Map (C2TM) **[3]**. It provides the traffic load statistics over 13k base stations that serve 450k users. The dataset was collected in a median-sized city in China during the period from August 19, 2012 to August 26, 2012. The cellular area covered is 50Km x 60Km. The authors of this dataset mentioned that they used the HTTP traffic at the city scale to extract the request-response records. A total number of 379 million HTTP records were generated during the collection week **[3].**

To measure the network load, it was mentioned in **[3]** that the connectivity to the internet on the user devices depended on the Radio Access Network, the Core Network, and the Public Network. The user device sends the data to the Base Transceiver Stations (BTSs) which transfers the data to a Base Station Controller (BSC). Then, through the Gb interface, the packets are transmitted to the Serving GPRS Nodes (SGSNs) and then sent to Gateway GPRS Support Nodes (GGSNs) which offer the connectivity between internal and external mobile networks. Finally, the Network Traffic Mining Platform (NTMP) receives the Bidirectional IP traffic from Gn interface. Inside the NTMP,

extraction and uploading of the HTTP logs to HDFS is executed by Deep Packet Inspection (DPI) for analysis **[3]**.

3.2 Dataset Basic Dimensions

The dataset covers the hourly traffic load on 13296 base stations for seven days or 168 hours. Georgian Calendar is used for the representation of time. For the location, the longitude and latitude of the connected base stations are reported in the dataset. Due to privacy reasons, the dataset providers mentioned that the locations used in this dataset are not the real locations of the base stations, instead they provided meshed locations.

The dataset consists of two files; a traffic file which consists of 1,625,680 rows and five columns and a topology file which consists of 13,296 rows and three columns. The first file includes the traffic load information and the second file includes the topology. The traffic file provides the statistics of the traffic load on each base station at each hour while the topology file shows the location of each base station.

According to city-cellular-traffic-map dataset repository, the description of each attribute is as follows [3]:

The traffic trace file:

- BS: the identification of every base station in the area covered
- Time_hour: the timestamp UNIX format
- Users: the number of users that use a specific base station during a specific hour
- Packets: the number of packets on a specific base station during a specific hour
- Bytes: the number of bytes on a specific base station during a specific hour

The topology file:

- BS: the identification of every base station in the area covered

- Lon: the longitude of a specific base station
- Lat: the latitude of a specific base station

3.3 Data Analysis

As mentioned earlier, the dataset consists of seven fields: bs, time_hour, users, packets, bytes, lon, and lat. The base stations IDs values range from 1 to 13,296, the number of users ranges from 0 to 413, the packets range from 0 to 794,585 and the bytes range from 0 to 114866512908.

Table 3.1 lists the initial features provided in the dataset and a sample of the values of each feature.

	bs	time_hour	users	packets	bytes	Lon	Lat
12303	100	2012-08-18 16:00:00	1	23	24642.0	110.984101	13.189041
12304	100	2012-08-18 17:00:00	1	1	5.0	110.984101	13.189041
12305	100	2012-08-18 18:00:00	3	657	891856.0	110.984101	13.189041
12306	100	2012-08-18 19:00:00	4	699	888934.0	110.984101	13.189041
12307	100	2012-08-18 20:00:00	2	833	1110008.0	110.984101	13.189041
12308	100	2012-08-18 21:00:00	4	49	54214.0	110.984101	13.189041
12309	100	2012-08-18 22:00:00	3	649	717248.0	110.984101	13.189041
12310	100	2012-08-18 23:00:00	4	50	48968.0	110.984101	13.189041
12311	100	2012-08-19 00:00:00	4	98	103292.0	110.984101	13.189041
12312	100	2012-08-19 01:00:00	10	887	3741328.0	110.984101	13.189041
12313	100	2012-08-19 02:00:00	7	2063	2627114.0	110.984101	13.189041
12314	100	2012-08-19 03:00:00	3	29	36137.0	110.984101	13.189041
12315	100	2012-08-19 04:00:00	3	13	1244.0	110.984101	13.189041
12316	100	2012-08-19 05:00:00	5	508	559320.0	110.984101	13.189041
12317	100	2012-08-19 06:00:00	10	3345	4353022.0	110.984101	13.189041
12318	100	2012-08-19 07:00:00	7	914	1144167.0	110.984101	13.189041
12319	100	2012-08-19 08:00:00	5	95	10615.0	110.984101	13.189041
12320	100	2012-08-19 09:00:00	5	586	763768.0	110.984101	13.189041
12321	100	2012-08-19 10:00:00	7	491	578746.0	110.984101	13.189041

Table 3. 1: Sample records from the dataset

By examining the correlation between the different numeric fields in the dataset, it seemed at first there is no clear linear correlation between the load on the base stations and any of the other fields. As illustrated in Figure 3.1 there is no correlation coefficient that exceeds 0.5.

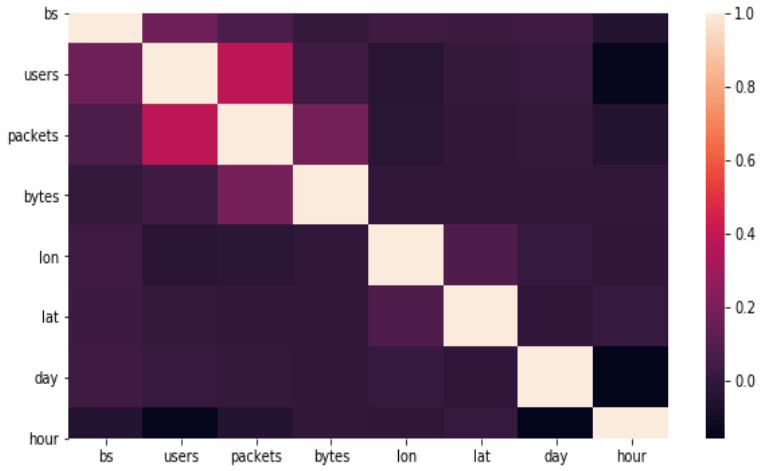


Figure 3.1: Attributes heatmap for Correlation Analysis

Since the actual location of the users is not given in this dataset, but rather the distribution of users, we are working under the assumption that areas with low numbers of users are rural areas and the areas with higher number of users are urban areas.

The distribution of the users is shown in Figure 3.2. The center area of the heatmap has a higher number of users and we consider this as a city, than the peripheral areas are rural.

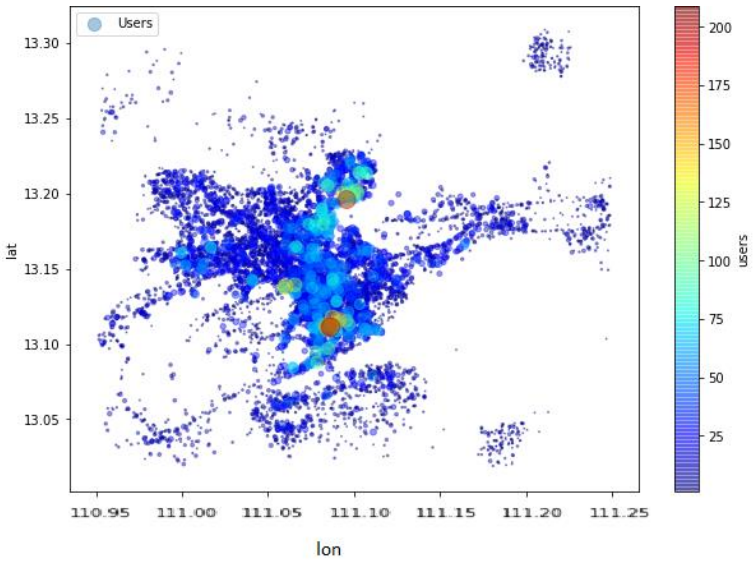


Figure 3.2: Users distribution heatmap

The following plots in Figure 3.3 show the average load on a few base stations during the seven-day period. We see that the load differs from one base station to the other, but it can be seen that there is a general upward trend between time 10 hr and 15 hr. Also the load decreases on most of the base stations after time 18 hr until 23 hr. That is expected, because the load during the rush hours and daytime is usually higher than the load at the late hours of the night.

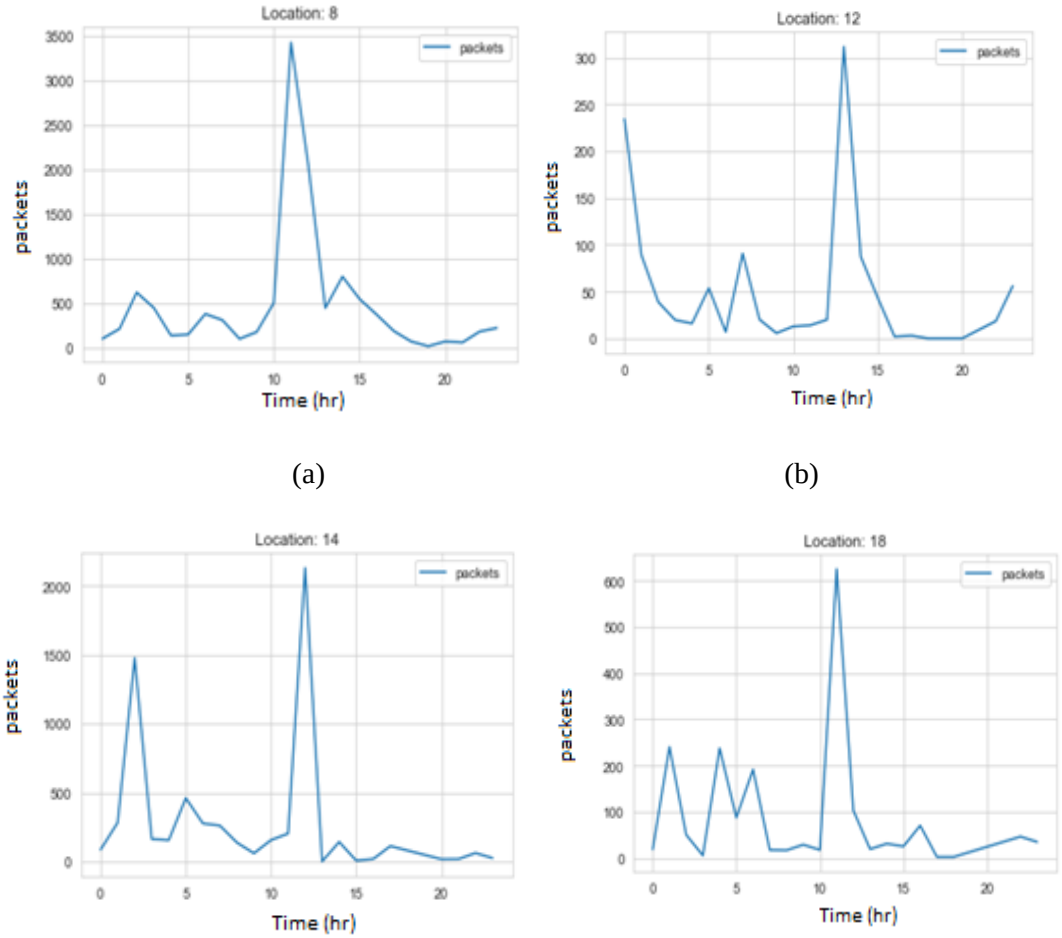


Figure 3.3: Average weekly load

Most of the time series forecasting techniques, require the time series to be stationary. Stationarity indicates that the statistical properties of the time series remain constant over time. Consequently, for a time series to be stationary, it needs both the standard deviation and the mean to not change

over the time. To test the stationarity of the load of the base stations, the rolling mean and the rolling standard deviation of the load of each base station over the seven days were calculated. It was observed that the mean and the standard deviation of the time series in the dataset are not constant over the time, which shows that the load of the base stations is not stationary. Therefore, the dataset required additional transformation to convert it to a stationary time series. Figure 3.4 shows the load of a sample the base station and displays the rolling standard deviation and the rolling mean through the week.

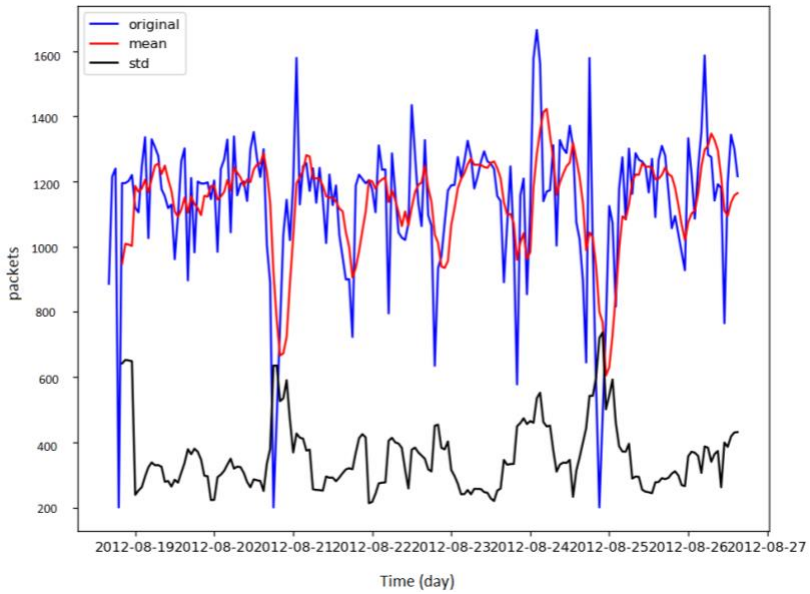


Figure 3.4: Standard deviation and mean vs the load

3.4 Dataset Limitations

Although this dataset provides the traffic load statistics for many base stations, it has a number of limitations that should be considered:

1. Relative Topology
2. Data Collection Duration
3. Granularity
4. Total users

5. Protocol Type

1. Relative Topology

The data provider of this dataset requested that the information concerning the real location and infrastructure be meshed. Hence, the location information is the relative topology not the real topology. As shown in Figure 3.5, the relative location of the base stations in the dataset is shown in an indiscriminate area. By having the relative locations instead of the actual ones, any additional information that depends on the location cannot be utilized, such as the weather or the events that happen at these locations. Moreover, the base stations that are located at rural areas and the base stations inside the city cannot be differentiated.

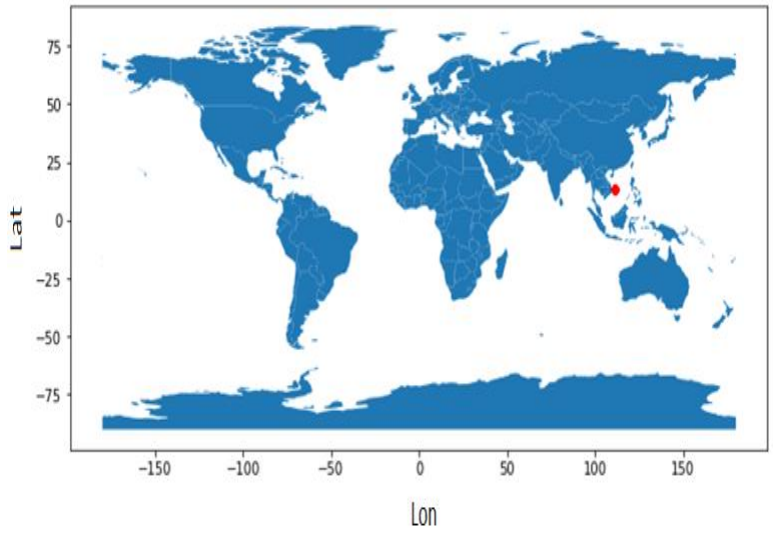


Figure 3. 5: Relative location of the area of interest of the base stations

2. Data Collection Duration

The data was collected for seven days from August 19, 2012 to August 26, 2012. That means that for each base station, only the traffic through this period is included. The given period is not long enough to deeply explore the behavior of the base stations. The behavior of the base stations could

change from one week to another, from one month to the next, and from season to season. However, given the short duration of data collection such changes would be missing.

3. Granularity

In this dataset, the load statistics are derived hourly at base-station granularity. With this granularity some of the information can be missed. The pattern of the load during the hour is not included in the dataset; the load can change from one second to the other, it can face many fluctuations during a single hour. Hence, the correct traffic load of the base stations on a smaller granularity, for example by one second or one minute, cannot be analyzed. The prediction capabilities are restricted to be on a larger scale of time.

4. Total users

The dataset only provides the total number of users connected to the given traffic load on the base stations each hour. There is no information about the usage of each user. The drawback of this is that the number of users will not give a correct indication about the load volume. There may be a large number of users with an average load, and a few users with a large load. That is because in most cases not all the users have the same usage. As shown in Figure 3.1, at record 12,309, three

users sent 649 packets while at record 12,315, there were also three users with only 13 packets sent.

If the individual usage was given, then the relation between each user and the traffic load would add more information and could improve the prediction performance since there would be a direct correlation between the users and their usage.

5. Protocol Type

The dataset providers used HTTP traffic to extract the request-response records. But HTTP underestimates the value of the real traffic load volume. Hence, the actual value is not reported which can affect the accuracy of the prediction.

3.5 Dataset Preparation

In order to be able to use the data properly for forecasting, a number of data preparation steps were applied on the dataset. They include:

1. Data Cleaning
2. Feature Engineering
3. Data Transformation

3.5.1 Data Cleaning

The first step in the data preparation process is data cleaning which has four steps:

a) Removing base stations with few records

b) Handling outliers

c) Defining missing data

d) Replacing NaNs

a) Removing base stations with few records

Although the dataset includes roughly 13k base stations, some of these base stations provide only few records in the whole dataset. A threshold was determined to represent the minimum number of records that the base station contributes in order to be considered, and the base stations that do not satisfy this threshold were removed.

First, the data were grouped by the base station ID, and for each base station the number of records that it has were counted. Then a threshold of 67 which represents a value of 40% of the maximum number of records (24 hours *7 days = 168 hours) was set. Only base stations with the number of records exceeding this threshold 67 are kept. The rest are removed.

b) Handling outliers

Outliers are the data points that are distinctly different from the rest of the data points in the dataset. The outliers may exist in the dataset because of noise or other random measurement errors that may happen during the recording or processing of the data. The presence of these outliers in the data causes inconsistencies. Inconsistency of the data can decrease the performance of the forecasting models. Hence, it is important to identify the outliers and remove them before applying other steps.

First, for the visualization of the outliers, two plots were used: Box plot and Scatter plot.

Box plot is used mainly in descriptive statistics. Figure 3.6 shows the basic components. The Box plot graphically represents numerical data collections using their quartiles. It shows the variability

lying outside the first (higher) and the third (lower) quartile, the Interquartile Range, as lines extending vertically from the box. Hence, the outliers can be shown as individual points [65].

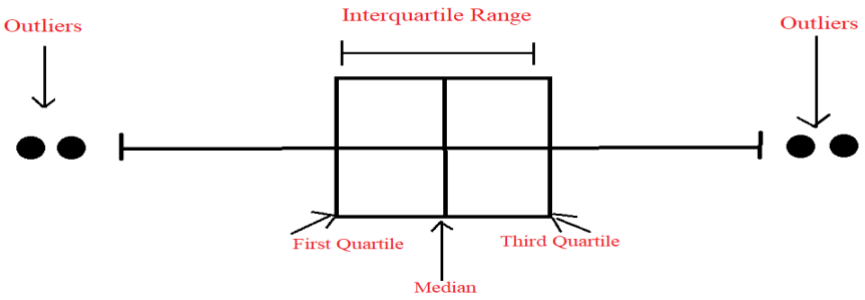


Figure 3. 6: Box plot basic diagram

Scatter plots uses Cartesian coordinates to show the values for a pair of points of a set of data. This data is presented as a group of points, where each point has a value that determines the horizontal axis and other value to determine the vertical axis [66].

The distribution of data before removing the outliers is shown in Figures 3.7 and 3.8 using the box plot and the scatter plot respectively. The Figures show that a significant number of outliers were found in the dataset.

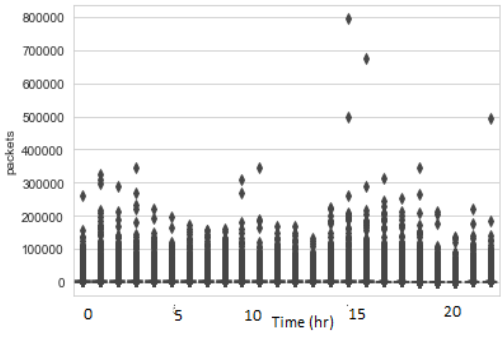


Figure 3. 7: Box plot with outliers

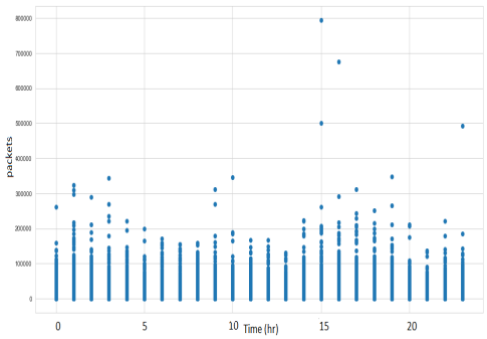


Figure 3. 8: Scatter plot with outliers

The Z-Score method is commonly used to remove outliers. Z-Score is used as the absolute value of the difference between the value of observation or data point and the mean of the observed or measured values. It can be calculated using the following equation:

$$z = \frac{x - \mu}{\sigma} \quad (3.\ 1)$$

where:

z represents the Z-score, x represents the data point value, μ represents the mean and σ represents the standard deviation.

Defining Outliers

The Z-score method was used to define the outliers in the data. The Z-score of all the data points were calculated. Then a threshold of three was determined. According to the empirical rule [67], 99.7% of the data points lie between +/-3 standard deviation. Hence, the interpretation of a datapoint that results in a Z-score greater than 3 is that this datapoint is different than 99.7% of the other data points.

For each data point, if the Z-score of the data point exceeded this threshold, the outlier value was considered as an outlier and initially replaced with a NaN which is used to represent the missing values.

c) Defining Missing data

As shown in Section 3.3, the time_hour column is not the original index of the data. Hence, it cannot be determined when a base station has a missing value. For this step, a list of the hourly date range for the period from August 19, 2012 to August 26, 2012 was created. Then the dataset was indexed by this list. Hence, it became easier to determine the missing values for each base station. The missing value was marked as a NaN value.

d) Replacing NaNs

Finally, for all the NaN values in the dataset that originated from the missing values or the outliers, we have examined multiple techniques to replace them. For instance, replacing the NaN value with the next value, the previous value and the median value of the previous and the next values. Based

on the results of the three techniques, the NaN values were chosen to be replaced with the value of the record at the previous time. As shown in the box plot Figure 3.9, and the scatter plot resulted in Figure 3.10, the number of outliers was significantly reduced.

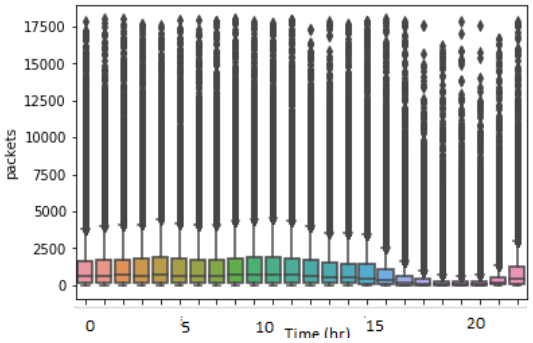


Figure 3. 9: Box plot without outliers

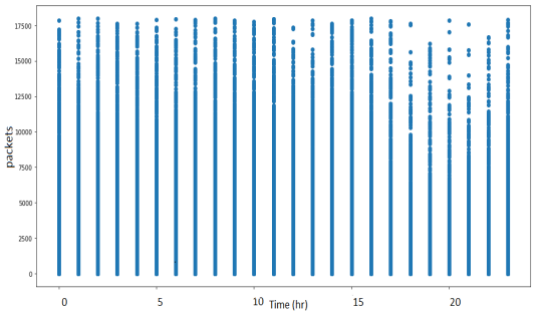


Figure 3. 10: Scatter plot without outliers

3.5.2 Feature Engineering

In the feature engineering step, some fields will be derived from the existing ones. For instance, the dataset provides one field that contains the hourly timestamp in UNIX epoch time, from which the hour, day, month fields were derived to test their individual effect on the base station load. A field, week_day, which determines if this day was a weekday (Monday - Friday) or weekend (Saturday and Sunday) was also added. Lastly, a field that combines the longitude and latitude together was derived; using the Haversine distance, which uses the longitude and latitude coordinates of two points lying on a sphere to calculate the great-circle distance between them. The origin point of the distance coordinates were assumed to be (φ, λ) = (0,0) where φ represents the latitude and λ represents the longitude. Then the Haversine distance between the base station location's coordinates and the origin point was measured. The distance is added as a new field, haversine_dist, to the dataset and calculated as [68]:

$$\text{Haversine Distance} = 2r \arcsin\left(\sqrt{\left(sin^2\left(\frac{\varphi_1-\varphi_2}{2}\right)\right) + \cos(\varphi_1)\cos(\varphi_2)sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \quad (3.2)$$

where

- φ_1 and λ_1 are the latitude and longitude of origin point
- φ_2 and λ_2 are the latitude and longitude of the base station

3.5.3 Data Transformation

The values in the data have very different scales. A normalization step was required to make all data fields have the same scale. Normalization was applied on all fields except for the hour field. To preserve its cyclical feature, it was transformed to two dimensions using sine and cosine transformation as follows:

- hour_sin=sin$(hour)$
- hour_cos=cos$(hour)$

As it was mentioned in Section 3.3, the data miss the stationarity characteristic. In order to convert the time series to stationary time series, differencing, seasonal differencing and log transformation techniques were examined. Differencing eliminates the level fluctuations in a time series which can lead to a decrease in trend and seasonality. Hence, the mean and the standard deviation can be kept stable **[37]**. The seasonal difference represents the difference between two consecutive observations lying in the same season **[30]**. In seasonal differencing the observation taken at a specific hour was subtracted from the same hour on the previous day. Log transformation is a common transformation technique that works by applying the log function on the time series to transfer each value to its logged one. After trying these three methods to convert the time series to stationary, log transformation was performed because it was more interpretable than the other two methods.

Chapter 4

Base Stations Clustering

This chapter discusses the following: the need for clustering and the problem it solves in our research, a comparison of the performance of three clustering techniques and the one chosen to cluster the base stations in the dataset, our methodology and our results using the different algorithms and similarity measures.

4.1 Introduction

Different base stations process different amounts of load traffic; while some base stations within a network may have a low load other base stations may have an extremely high load and that any given time the opposite could be true. As shown in Figure 4.1, these are the average traffic loads of four different base stations during the week, each base station has a completely different traffic load than the other.

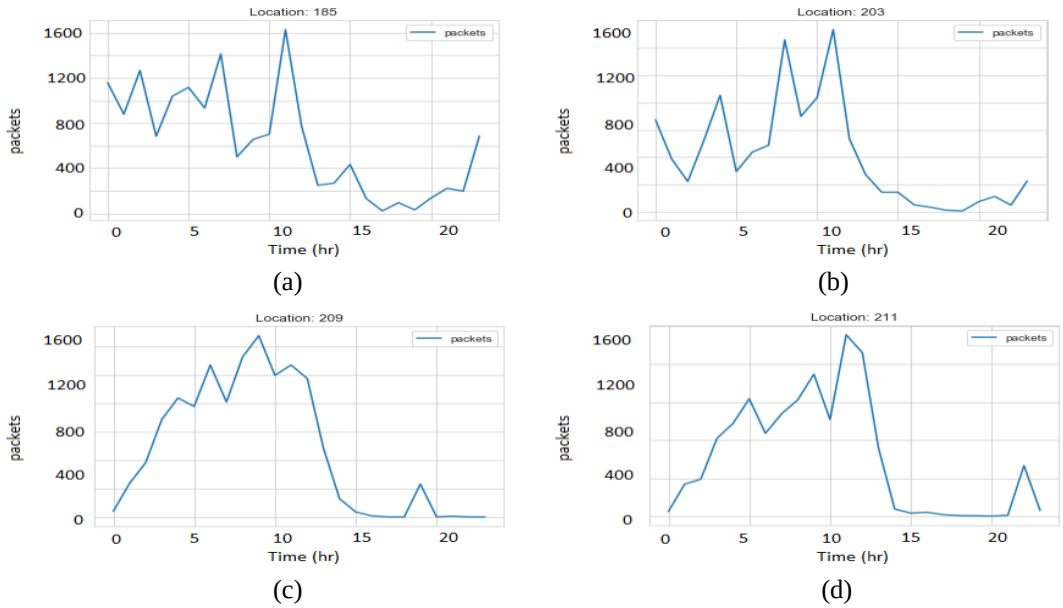


Figure 4.1: Different traffic load at base stations

For time series forecasting, the larger the training data the better performance one can achieve. But, in case of the data encountering different behaviors, this may not improve the performance and it can also degrade it. Hence, the base stations that have the similar traffic loads need to be trained together. To solve this problem, clustering was used in this project to group the base stations with similar traffic loads together.

In the clustering process, the similarities between the data is extracted based on the data features. Clustering time series data differs from clustering static data in the aspect of comprising dynamic values for the features [69]. Most of the traditional clustering techniques fail to provide satisfying results in clustering time series data because they are designed for extracting similarities from static features [70].

4.2 Clustering Techniques

Clustering can be done according to several properties. In this research, two properties were used to perform the clustering.

1. Spatial Clustering, which focuses on clustering the time series according to their location.
2. Time Series Clustering, which focuses on clustering the time series according to their behavior.

4.2.1 Spatial Clustering

Spatial clustering assumes that adjacent base stations may contend with the same behavior. Figure 4.2 shows the locations of all the base stations. The map was divided into small grids and each grid represented a cluster. Base stations located in same grid were assigned to the grid cluster number.

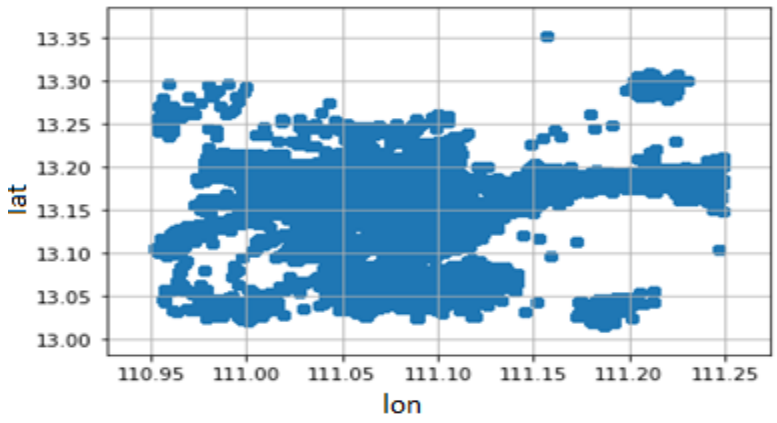


Figure 4.2: Base stations Locations

To test the performance of this technique, the behavior of base stations that belong to the same location were examined. However, as shown in Figure 4.3 and Figure 4.4, base stations at the same location have very different traffic loads. As a result, it can be noted that the behavior similarity of a group of base stations was not dependent on their location. Therefore, this technique does not provide convincing results and another approach was needed to perform the clustering process.

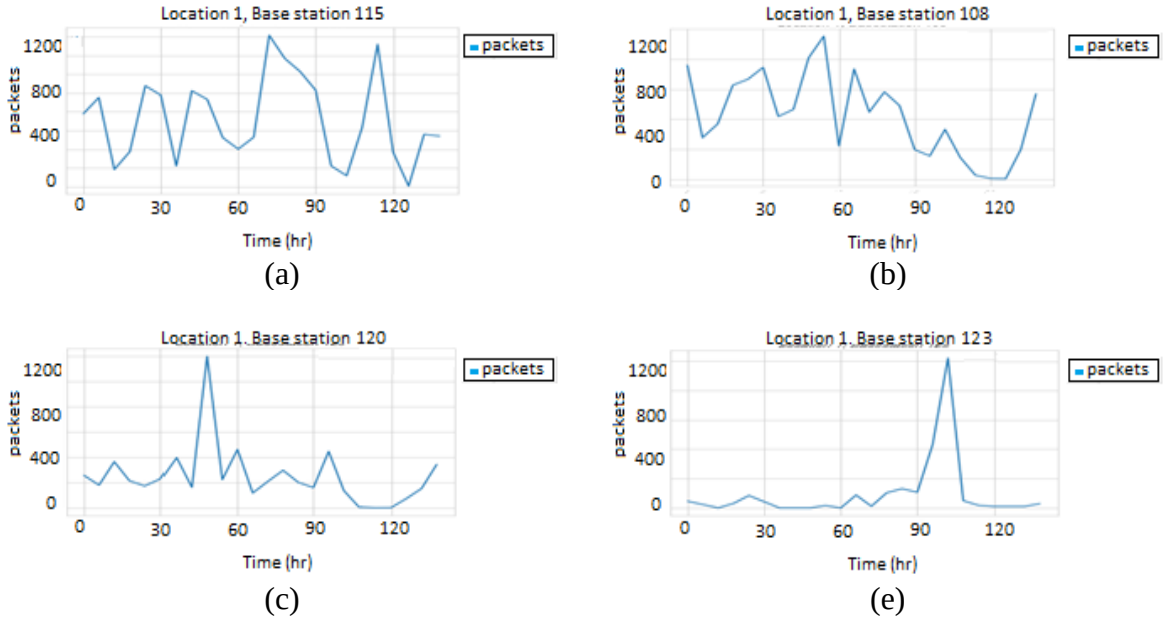


Figure 4.3: Traffic load in Location #1

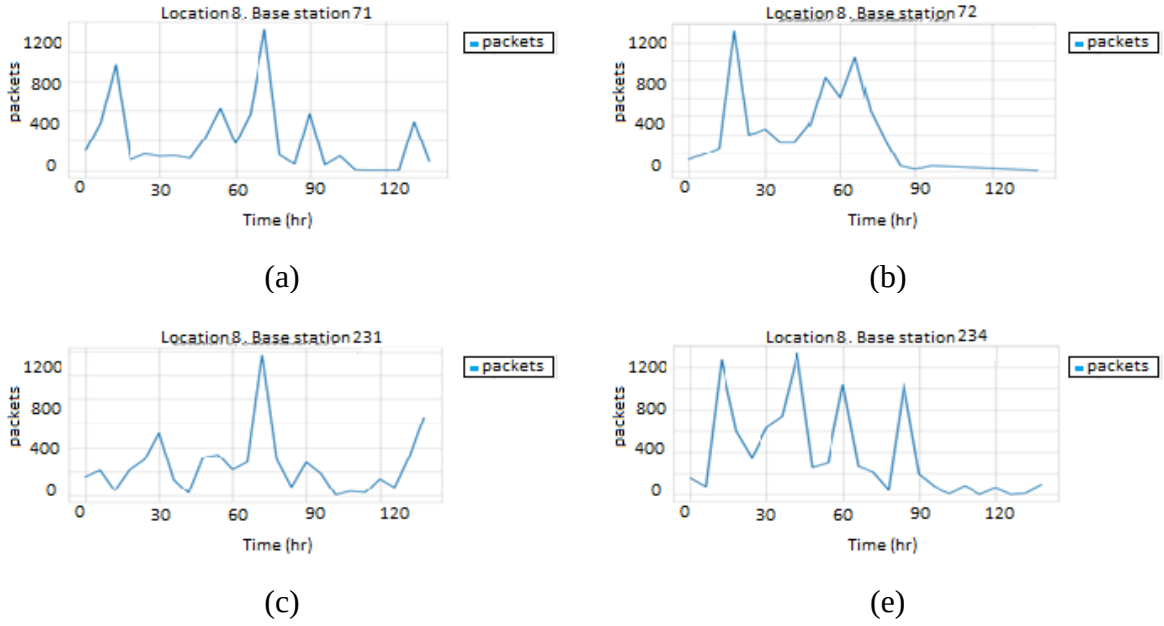


Figure 4.4: Traffic Load in Location #8

4.2.2 Time Series Clustering (Clustering by behavior)

Instead of the location, time series clustering groups basestations according to their behavior. For instance, as shown in Figure 4.5, although base station 19 and base station 7, base station 23 and base station 24 are not necessarily neighbors, they share similar behaviors. Hence, base station 19 should be clustered with base station 7 in cluster C1 and base station 23 should be clustered with base station 24 in cluster C2

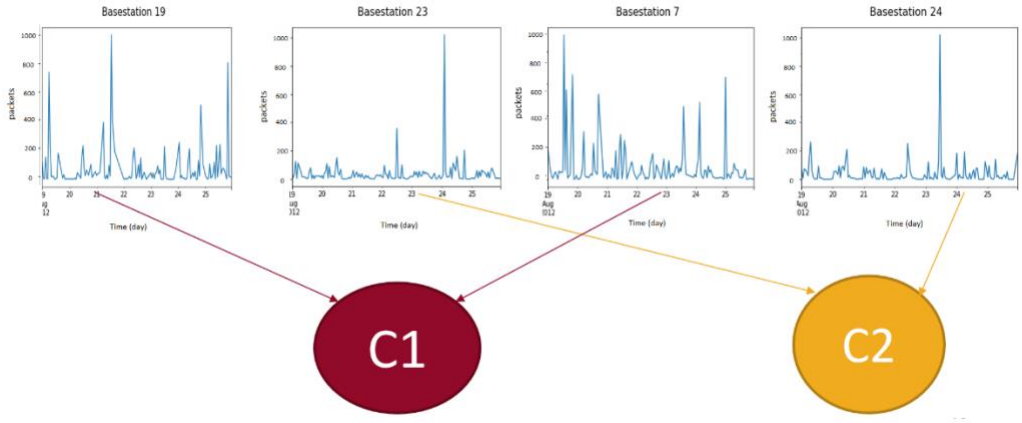


Figure 4.5: Time Series Clustering

4.3 Clustering Methodology

The first task needed to cluster the base stations by behavior, is defining the metric by which the similarity of the base stations is assessed. The behavior of a base station can be treated as a time series problem. Several distance measures can be used to measure the similarity between two time series. The second task required after defining the similarity metric is determining the clustering algorithm by which the base stations are clustered. In this thesis, multiple combinations of the clustering algorithms and similarity metrics discussed in Chapter 2, were examined. The tslearn package, Global Alignment Kernel KMeans, K-Shape and TimeSeriesKMeans were used with both the Euclidean distance and Dynamic Time Warping metrics. With the dtwclust package, partitional clustering, hierarchal clustering and K-shape clustering were also examined with the mentioned distance metrics. Also, Mean and median, Partition around medoids and Shape extraction centroids types were used for initialization.

The mentioned clustering methodology packages needs the data to be formatted as an array, for first step, the dataset was converted from its original form to a multidimensional array where each array represents the data of a single base station. Also, the data was examined twice, in normalized and unnormalized forms

4.4 Results

Several metrics were used for error calculation. Let C represent the Clusters vector, X represents the dataset, k represents the number of clusters and d represents the distance between the objects.

Silhouette index (SIL): This measures the distance between the time series and the centroid of the cluster they belong to compared to the centroids of the other clusters. The Silhouette index equation is defined as [71]:

$$\text{SIL(C)} = \frac{1}{N} \sum_{c_k \epsilon C} \sum_{xi \in c_k} \frac{b(x_i, c_k) - a(x_i, c_k)}{max\{a(x_i, c_k), b(x_i, c_k)\}} \quad (4.1)$$

$$\text{a}\,(x_i, c_k) = \frac{1}{|c_k|} \sum_{x_i \epsilon c_k} d\_e\,(xi, xj) \tag{4.2}$$

$$\text{b}x_i, c_k) = min_{c_l \epsilon C \setminus c_k} \{ \frac{1}{|c_l|} \sum_{x_i \epsilon c_l} d\_e\,(xi, xj\,)\} \tag{4.3}$$

- **Davies-Bouldin index (DB):** This evaluates the inner cluster similarity against the outer cluster similarity. The Davis-Bouldin index equation is [71]:

$$\text{DB(C)} == \frac{1}{K} \sum_{c_k \epsilon C} max_{c_l \epsilon C \setminus c_k} \{ \frac{S(c_k\,), S(c_l)}{d\_e(c_k\,, c_l)} \} \tag{4.4}$$

$$\text{S}\,(c_k) = \frac{1}{|c_k|} \sum_{x_i \epsilon c_k} d\_e\,(xi, c_k) \tag{4.5}$$

- **Modified Davies-Bouldin index (DB*):** This uses the same concept as of the DB but provides higher speed. It can be calculated as follows [71]:

$$\text{DB*(C)} = \frac{1}{K} \sum_{c_k \epsilon C} \frac{max_{c_l \epsilon C \setminus c_k} \{S(c_k\,)\, S(c_l)\}}{min_{c_l \epsilon C \setminus c_k} \{d\_e(c_k\,, c_l)\}} \tag{4.6}$$

$$\text{S(C)} = \frac{1}{N} \sum_{c_k \epsilon C} \sum_{xi, xj \epsilon c_k} d\_e\,(xi, xj) \tag{4.7}$$

- **Dunn index (D):** This calculates the means of the distances between the time series in one cluster against the means of the distance of the other clusters. Its equation is [71]:

$$D(c) = \frac{min_{c_k \epsilon C} \left\{ min_{c_l \epsilon C \setminus c_k} \{\delta(c_k, c_l)\} \right\}}{max_{c_k \epsilon C} \{\Delta(c_k)\}} \tag{4.8}$$

$$\delta(c_k, c_l) = min_{x_i \epsilon Ck} min_{x_j \epsilon Cl} \{d\_e\,(xi, xj\,)\} \tag{4.9}$$

$$\Delta(c_k) = max_{x_i x_j \epsilon Ck} \{d\_e\,(xi, xj\,)\} \tag{4.10}$$

- **COP index (COP):** This uses the distance of the series in a cluster to its centroid and the compares the median of the series to the maximum distance COP index is calculated as follows [71]:

$$\text{COP(C)} = \frac{1}{K} \sum_{c_k \epsilon C} |c_k| \frac{\frac{1}{|c_k|} \sum_{x_{i,} \epsilon c_k} d\_e\,(xi, c_k)}{min_{x_i \notin Ck} max_{x_j \epsilon Cl} \{d\_e\,(xi, xj\,)\}} \tag{4.11}$$

Different numbers of clusters were attempted in this research, but the highest scores were produced using five, eight and ten as the number of clusters, we focused on presenting the results that were produced using all the explored methods. Regarding the centroid selection technique, no significant improvement was found using the partition around medoid technique. The results shown consider the median technique because it is faster than the partition around medoid.

To test the effect of the normalization on clustering, the techniques used were examined using both normalized and non-normalized data.

4.4.1 Analysis

In Table 4.1 and Table 4.2 in the comparison between the algorithms that were used using the tslearn model which are: KShape, Global Alignment Kernel and TimeSeriesKMeans algorithms were presented. The TimeSeriesKMeans produced better results in both forms of normalized and unnormalized data than the KShape and the Global Alignment Kernel algorithms. A slight increase in the scores can be noted in case of using eight clusters instead of ten clusters. The best results were produced using five as the number of the clusters. By comparing the results of the normalized data against the non-normalized data, it can be shown that they produce comparable results with a slight increase in the score in case of the normalized data. Regarding the distance metric, it is shown that using the Euclidean distance resulted in lower scores for both the DTW and Soft-DTW in all of the different variations. However, between the dtw and soft-dtw, the results are very similar, and no noticeable change can be found. With regard to the computation speed, Euclidian distance was very fast compared to both the soft-dtw and dtw which both required much more execution time and that is due to the high time complexity of the dtw algorithm which is O(N^2) compared to O(log(n)) for the Euclidian distance algorithm [72] [73] [74].

Moving to the algorithms used by the dtwclust library, which are partitional clustering and the hierarchical clustering. By comparing the SIL, DB, DBSTAR, D and COP scores, it is shown in Table 4.3, that the performance of the partitional clustering exceeds the hierarchical clustering. In addition, for the distance metrics and data normalization, it is shown that the same conclusions can be observed from the resulted values. Normalized data with five clusters produced better results than the other selections. Likewise, the Euclidian distance produced the lowest scores compared to other distance metrics.

Number of Clusters	Silhouette Score KShape Algorithm		Silhouette Score Global Alignment Kernel algorithm	
5	**-0.05**	**-0.062**	**-0.028**	**-0.024**
8	**-0.05**	**-0.051**	**-0.032**	**-0.028**
10	**-0.06**	**00.044**	**-0.047**	**-0.039**

Table 4.1: Results of KShape and Global Alignment Kernel Algorithms

	10 clusters		8 clusters		5 clusters	
Euclidean	**0.0674**	**0.059**	**0.068**	**0.167**	**0.23**	**0.21**
DTW	**0.093**	**0.161**	**0.0957**	**0.188**	**0.39**	**0.35**
SoftDTW	**0.052**	**0.17**	**0.214**	**0.194**	**0.41**	**0.38**

Table 4.2: Results of TimeSeriesKMeans algorithm for Normalized/Unnormalized data

	SIL		DB		DBStar		D		COP	
PC,10c, dtw	**-0.03**	**-0.02**	**1.07**	**1.85**	**1.10**	**2.032**	**0.62**	**0.29**	**0.70**	**0.77**
PC,10c, sdtw	**-0.021**	**-0.022**	**1.11**	**1.90**	**1.15**	**2.037**	**0.67**	**0.34**	**0.72**	**0.81**
PC,10c, euclidean	**0.03**	**-0.015**	**1.38**	**2.055**	**1.43**	**2.294**	**0.70**	**0.35**	**0.74**	**0.70**
PC,10c, sbd	**-0.008**	**-0.002**	**1.72**	**1.4**	**1.79**	**2.015**	**0.28**	**0.37**	**0.64**	**0.61**
HC,10c, euclidean	**0.03**	**0.05**	**1.26**	**1.32**	**1.12**	**1.18**	**0.59**	**0.68**	**0.66**	**0.70**
PC,8clusters, dtw	**0.14**	**0.1**	**1.311**	**1.011**	**1.393**	**1.013**	**0.44**	**0.57**	**0.70**	**0.68**
PC,8c, sdtw	**0.1**	**0.18**	**1.5**	**1.53**	**1.82**	**1.56**	**0.27**	**0.38**	**0.57**	**0.52**
PC,8c, eucledian	**0.04**	**0.06**	**1.44**	**1.69**	**1.48**	**1.74**	**0.72**	**0.65**	**0.74**	**0.71**
PC,8clusters, sbd	**-0.01**	**0.001**	**1.86**	**1.7**	**1.89**	**1.92**	**0.41**	**0.34**	**0.69**	**0.58**
HC,8c, euclidean	**0.07**	**0.06**	**1.16**	**1.21**	**1.22**	**1.22**	**0.73**	**0.73**	**0.75**	**0.75**
PC,5c, dtw	**0.34**	**0.31**	**1.47**	**1.024**	**1.582**	**1.019**	**0.44**	**0.77**	**0.75**	**0.78**
PC,5c, sdtw	**0.39**	**0.33**	**1.64**	**1.73**	**1.92**	**1.76**	**0.29**	**0.58**	**0.62**	**0.62**
PC,5c, eucledian	**0.16**	**0.17**	**1.49**	**1.89**	**1.54**	**1.94**	**0.76**	**0.75**	**0.81**	**0.81**
PC,5c, sbd	**0.08**	**0.05**	**1.86**	**1.9**	**1.93**	**2.2**	**0.43**	**0.54**	**0.83**	**0.78**
HC,5c, euclidean	**0.11**	**0.09**	**1.23**	**1.31**	**1.47**	**1.42**	**0.75**	**0.93**	**0.85**	**0.85**

Table 4.3 Results of: cvi scores for Normalized/ Unnormalized Data: PC stands for Partitional Clustering, HC stands for Hierarchical Clustering, c stands for clusters number

4.4.2 Final Algorithm and Metrics

As discussed above, the partitional clustering method produced the best results. To determine the tool to be used, the silhouette score was considered as the final metric to be used. The silhouette score of the partitional clustering in the tslearn library by the TimeSeriesKMeans algorithm was higher than that of the partitional clustering using the dtwclust library. Therefore, tslearn was chosen over dtwclust. The TimeSeriesKMeans was chosen to be used as the algorithm. Soft-Dynamic Time Warping was used as the distance metric and the number of clusters was set to five. After settling on the clustering technique, a new column was added to the dataset that represents the cluster number for each base station and the base stations were grouped by these cluster numbers.

4.4.3 Final Clusters Analysis

Five clusters were created. The number of base stations in each cluster was as follows:

Cluster Id	Base stations count
0	129
1	28
2	1308
3	168
4	438

Table 4.4: Number of Base stations in each cluster

Figure 4.6 shows the representation of the centroid among the other data, the dashed lines represent the centroid of the cluster and the series that belong to this cluster are plotted on top of it. The color provides an indication of the number of the base stations in each cluster. The second cluster has a different color than other clusters because it includes a significantly higher number of base stations.

It is observed that the maximum load in each cluster differs from others. While the maximum number of packets in the third and fourth cluster is close to 1600, the fifth cluster can only reach 800. It can also be noted that some centroids give a better representation of the clusters than the others. For instance, in the first, third, and fifth clusters the centroid covers most of the trends found in the series but in the second cluster the centroid cannot give adequate information about the cluster. The reason for this can be attributed to the similarity of the behavior in each cluster. When the similarity is higher the average of the series will be close to the most series in the cluster. If the similarity is lower, the average will not be an accurate representation for the series in the cluster.

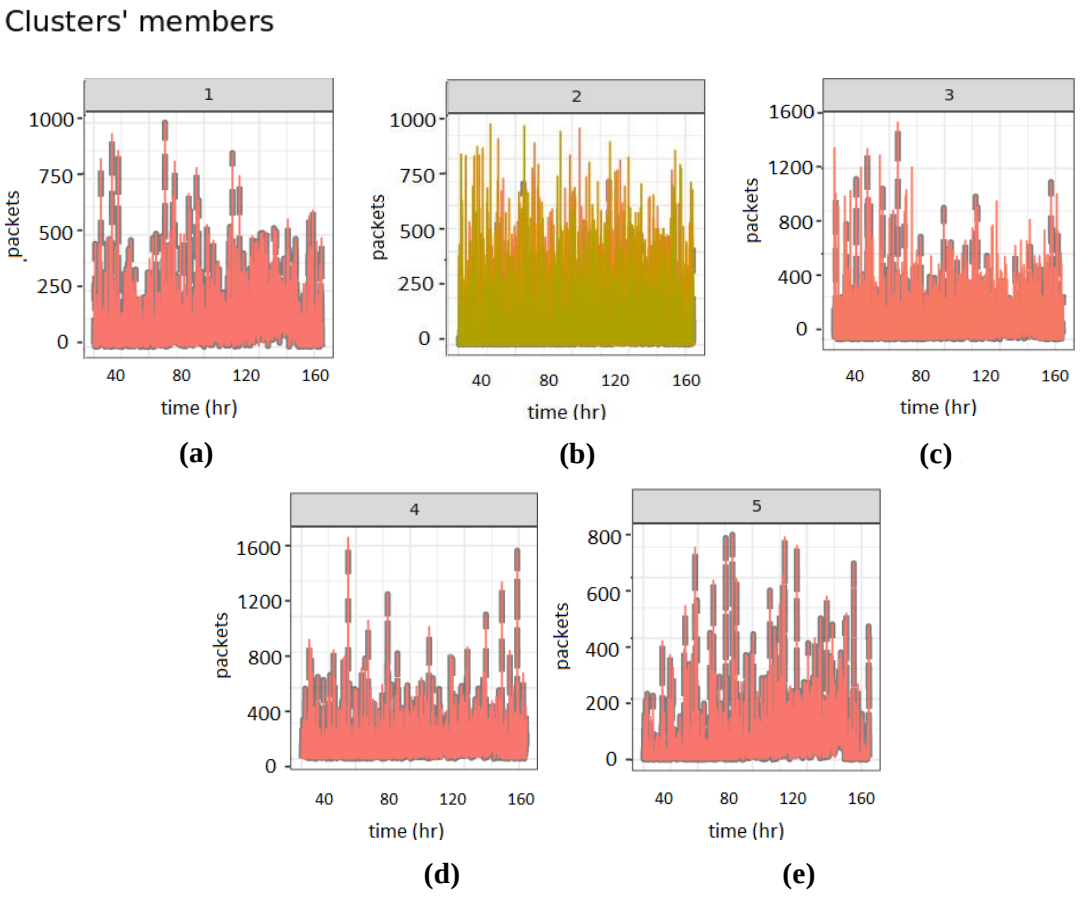


Figure 4.6: Final Cluster Visualization

Chapter 5

Forecasting Results and Discussion

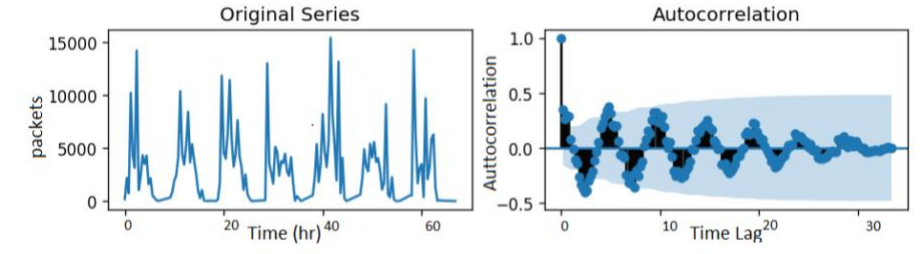
In this chapter, the results of each model from the discussed forecasting models are presented, and the setup of each model is described. For each model the Root Mean Squared Error, which is a measure of the standard deviation of the errors in the prediction, was used to determine the error produced by each model. The focus of the results was on the tests made on a one step ahead forecasting. Lastly, a discussion about the performance of the different models and a comparison between them took place. Starting with Section 5.1, the statistical model results are presented. Then in Sections 5.2 and 5.3 the machine learning and the deep learning results are shown. After that the online tools, DeepAR and Prophet, results are reviewed in Sections 5.4 and 5.5. At the end, a discussion about the performance of all models is in Section 5.6.

5.1 Performance Based on Statistical Models

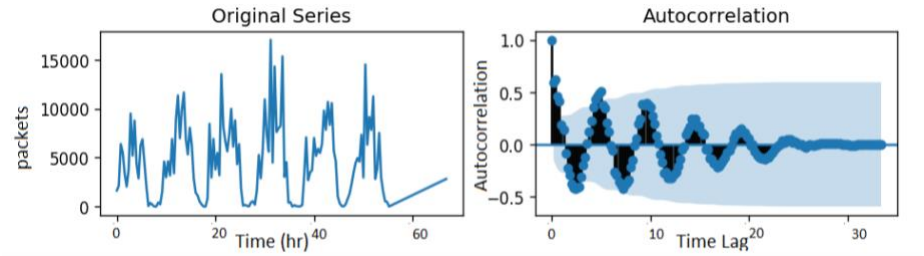
a) ARIMA Model

ARIMA model works with univariate time series. Therefore, for testing the ARIMA and SARIMA models, each base station was trained and tested independently. For each base station, 70% of the data were used for training and 30% for testing. In this section, the results of a representative base station from each cluster is demonstrated and discussed.

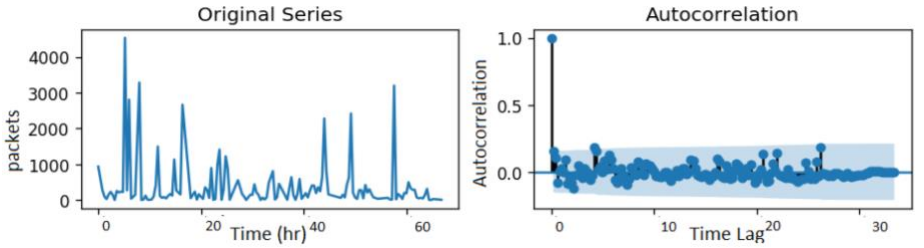
As mentioned in Chapter 2, stationarity is a must for statistical models forecasting. Hence, the transformation methods were applied on the data to make it stationary. One characteristic for stationary data is that the values in its Autocorrelation Function (ACF) plot would be converging towards zero [75]. As shown in the ACF plots of some sample base stations in the dataset presented in Figure 5.1, the data is stationary and does not require further differencing.



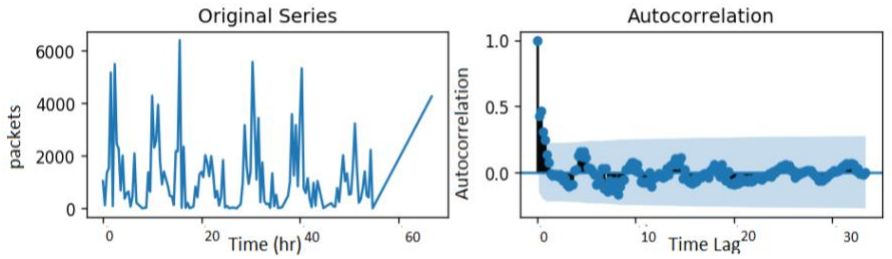
Cluster 0



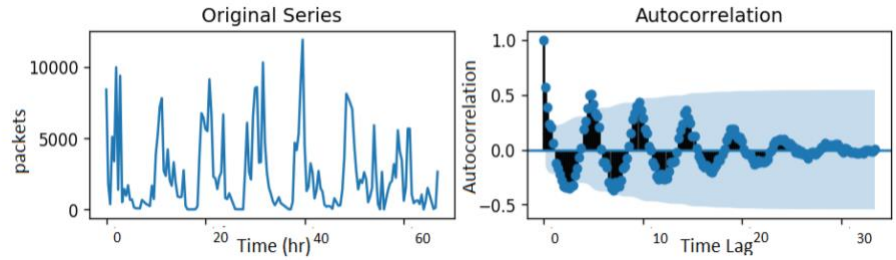
Cluster 1



Cluster 2



Cluster 3

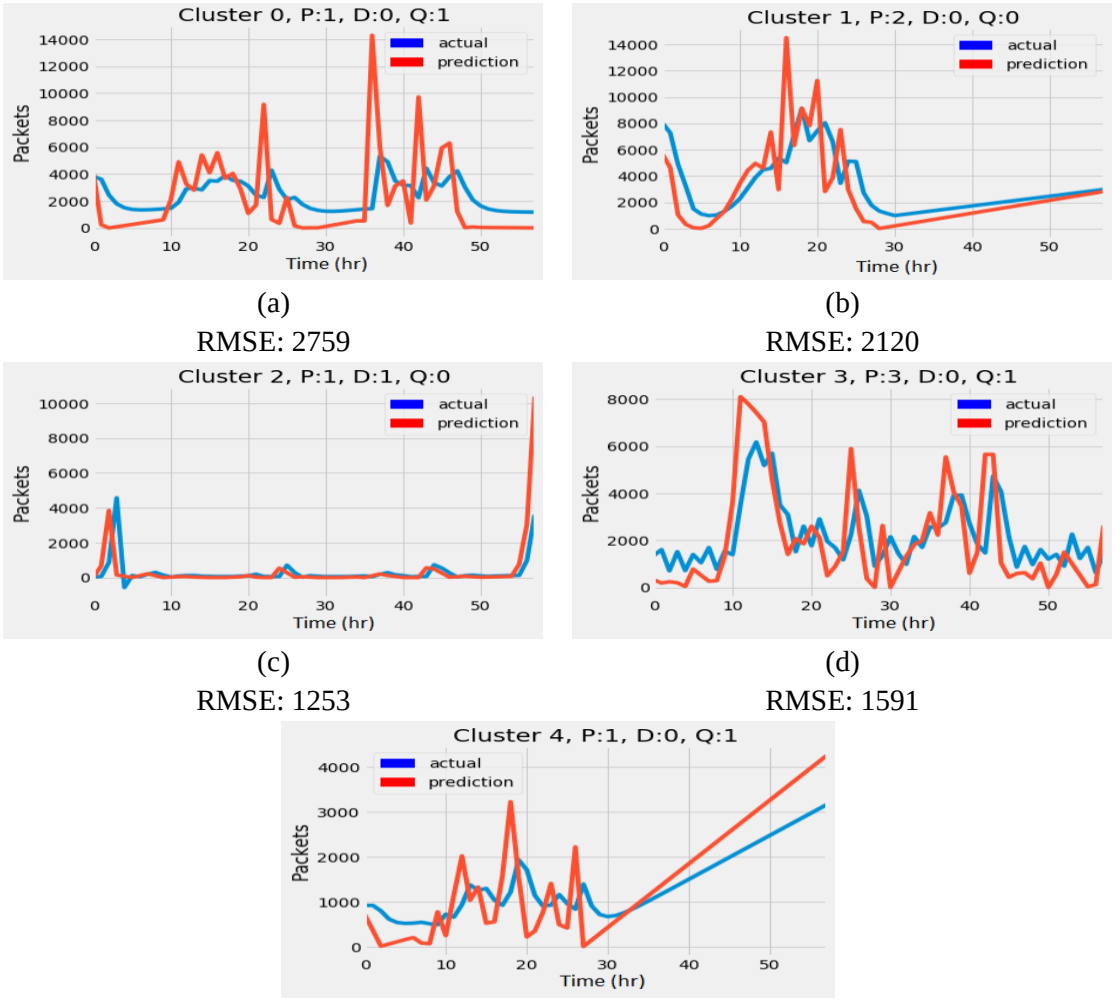


Cluster 4

Figure 5.1: Clusters Autocorrelations

Results Based on ARIMA

For choosing the ARIMA model hyper parameters, p, d, and q, grid search was applied. The values that produced the least root mean squared error were chosen. The search was on the range of zero to three for the three hyper parameters. The following subfigures in Figure 5.2 depict the prediction vs the actual value of each base station along with the p, d, and q values.



(a)
RMSE: 2759

(b)
RMSE: 2120

(c)
RMSE: 1253

(d)
RMSE: 1591

(e)
RMSE: 2759

Figure 5.2: ARIMA Results

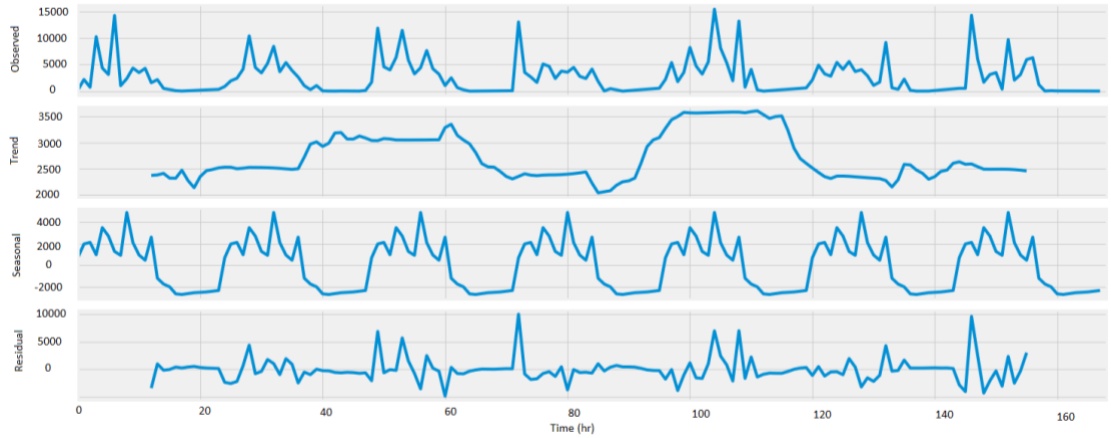
Figure 5.2 shows that the ARIMA model resulted in high RMSE for all clusters. However, it is noted from the subfigures that the ARIMA predictions usually follow the same patterns that are followed by the actual time series. In further detail, if there is an increase in the future load, the

ARIMA model predicts the increase but does not predict the value of the increase correctly. This can be beneficial in a classification version of the problem, where the ARIMA model can be utilized to predict the future load trend.

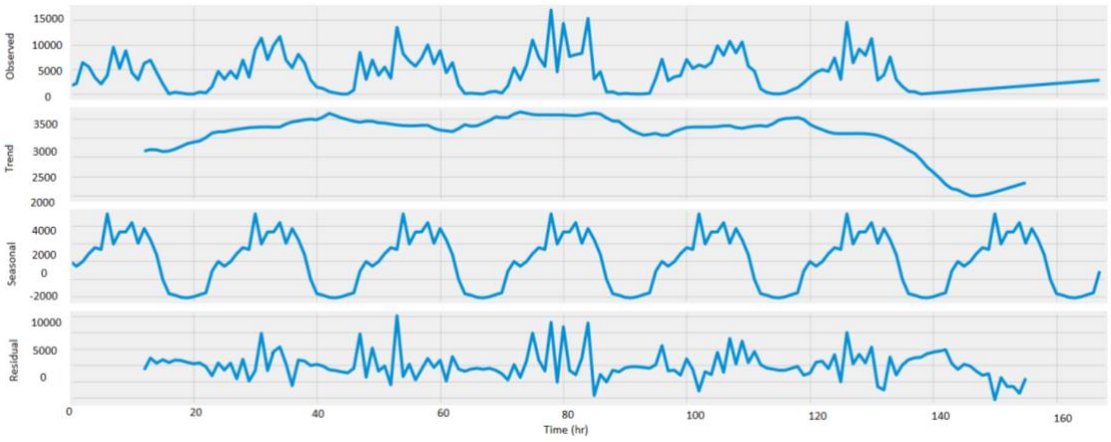
b) SARIMA Model

SARIMA is used to handle the seasonality in the time series. Figure 5.3 shows a sample of the decomposition of base stations belonging to each cluster. The seasonal component in each figure shows that there is a seasonality in the data. Hence, SARIMA was applied to explore the performance of handling this seasonality.

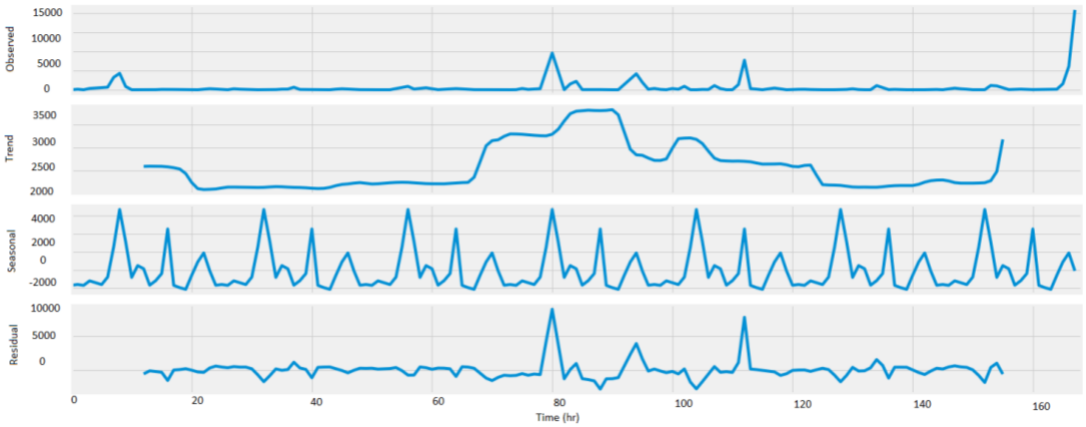
(a) Decomposition of Cluster0



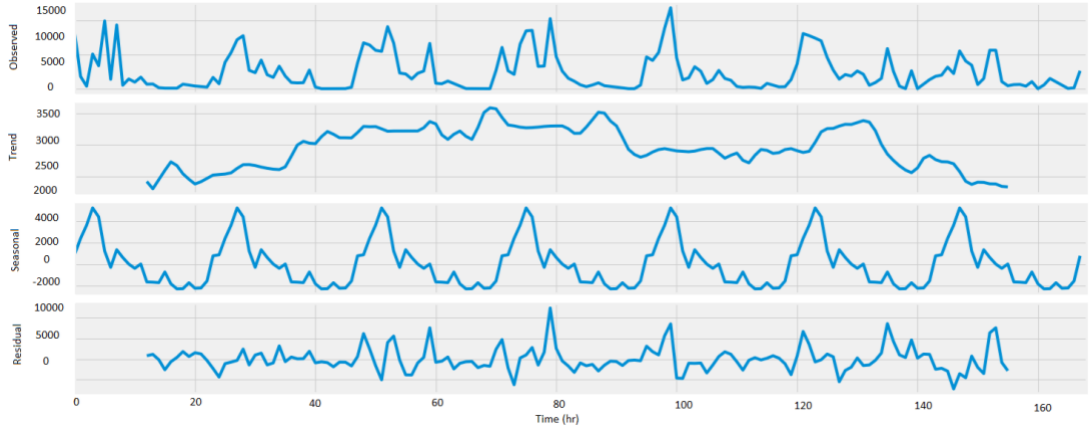
(b) Decomposition of Cluster1



(c) Decomposition of Cluster2



(d) Decomposition of Cluster3



(e) Decomposition of Cluster4

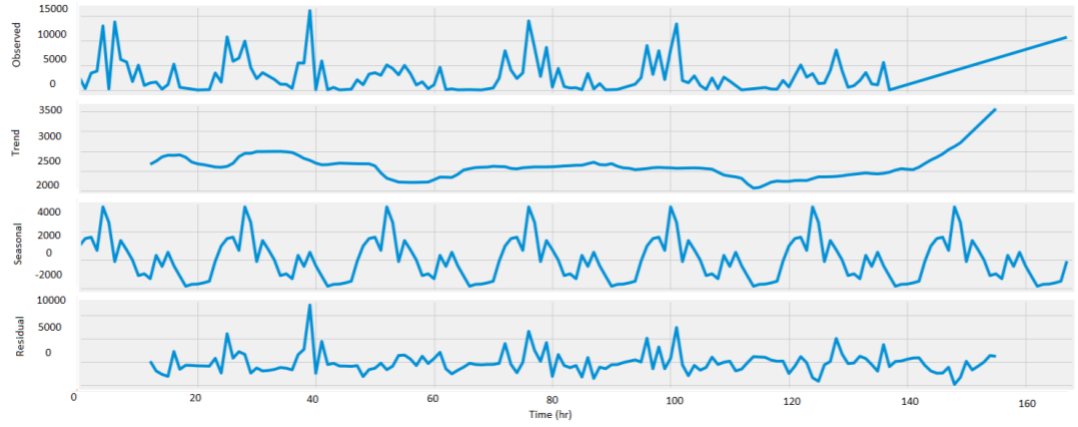
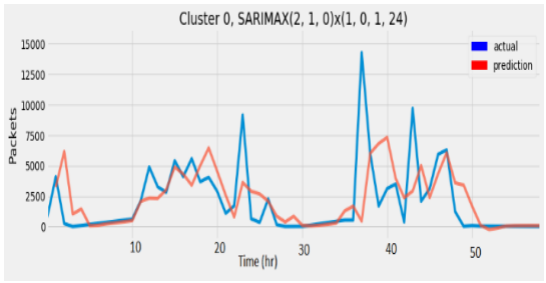


Figure 5.3: Clusters Decomposition

Results Based on SARIMA

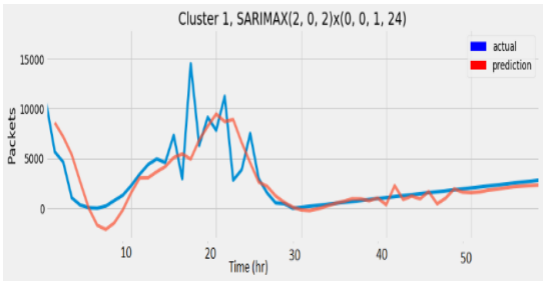
As in the ARIMA model, grid search was applied to choose the hyperparameters of the models, 70% of the data were used for training and 30% for testing. and the root mean squared error was chosen as the error metric. As mentioned in Chapter 4, SARIMA requires three more hyperparameters than the ARIMA which are the P, D, and Q.

Figure 5.4 plots the prediction versus the actual value of a representative base station of each cluster along with the p, d,q, P, D, and Q values.



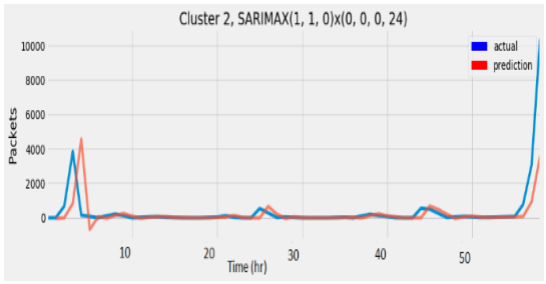
(a)

RMSE: 1013



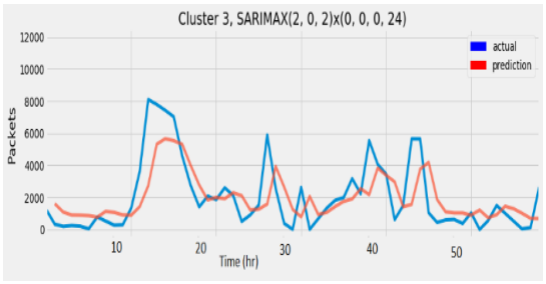
(b)

RMSE: 1215



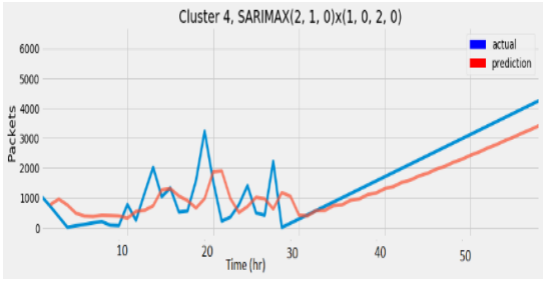
(c)

RMSE: 3840



(d)

RMSE: 1075



(e)

RMSE: 1038

Figure 5.4: SARIMA Results

The RMSE resulted from the SARIMA model was significantly lower than the RMSE resulted from the ARIMA model. This is probably because of the seasonality in the data that can be handled by the SARIMA model. However, the RMSE is still not low, which indicates that, as the ARIMA model, SARIMA can perform better in predicting if the load is increasing or decreasing than predicting the actual value of the future load.

5.2 Performance Based on Machine Learning

5.2.1 Machine Learning Data Preprocessing:

To train the machine learning models, a new column was added to the dataset that represents the traffic load of each base station but shifted by the number of steps defined for prediction. This future value was considered as the target for training the models. Each cluster was trained using a different model than the other cluster. Hyperparameter tuning using grid search was used for choosing the hyperparameters of each algorithm. For each model 75% of the data were used for training and 25% were used for testing. In the following sections, the setup of each algorithm is shown and followed by the results of each cluster.

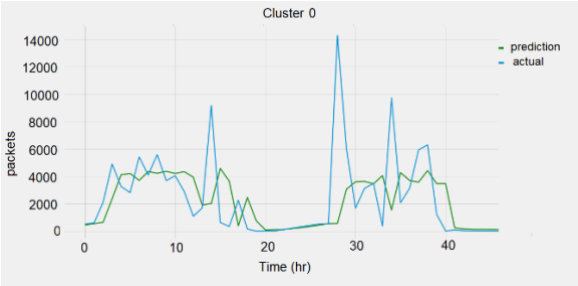
5.2.2 Performance Based on SVM

For preparing the SVM model, the following values of the hyperparameters are defined [76]:

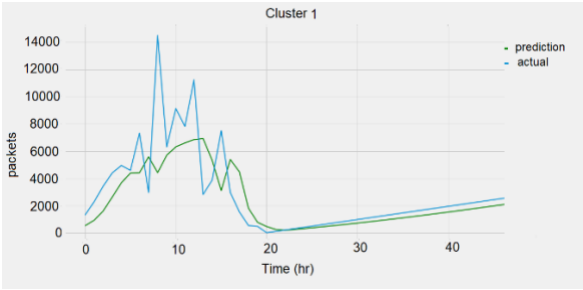
- **Kernel:** defines the type of kernel that the SVM model uses.
- **Gamma:** defines the coefficient of the kernel.
- **Epsilon:** defines the allowable range of error during the training process.

The grid search included three types of kernels: linear, polynomial and Radial Basis Function (RBF) kernels. Between the three choices, the RBF kernel performed better in all models. The kernel was set to RBF in all trained models. For the epsilon, three values were explored: 0.1, 0.01 and 0.5. Then, for the gamma the search was on 0.1, 0.01 and 0.001. When the value of gamma is

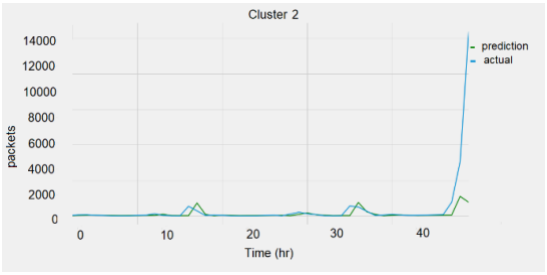
small, the constraints on the model will increase and it becomes harder to deal with nonlinear or complex data and vice versa. In Figure 5.5, the results of each cluster are presented. Under each plot, the hyperparameters values that gave the least RMSE are presented.



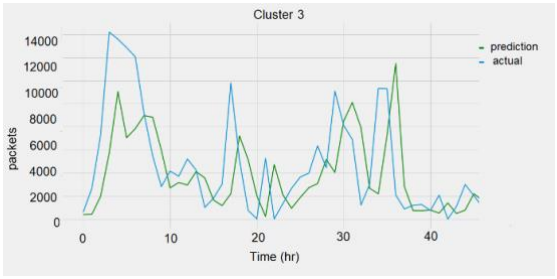
(a)
Gamma:0.001
Epsilon:0.1, RMSE: 3118



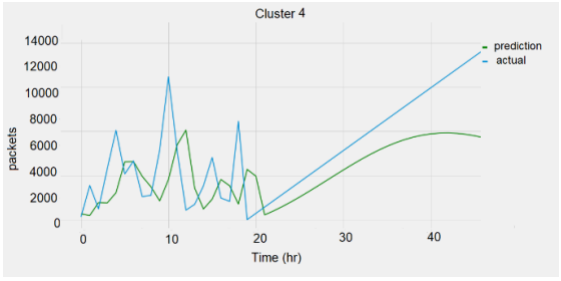
(b)
Gamma:0.001
Epsilon:0.1, RMSE: 2584



(c)
Gamma: 0.1
Epsilon:0.1, RMSE: 1474



(d)
Gamma:0.001
Epsilon:0.1, RMSE: 2326



(e)
Gamma:0.001
Epsilon:0.1, RMSE: 1587

Figure 5.5: SVM Results

The results above were worse than the results provided by SARIMA. SARIMA provided lower RMSE in most clusters' models except for the third cluster. This indicates that in comparison to the statistical models, the SVM may not be ideal in some time series forecasting problems.

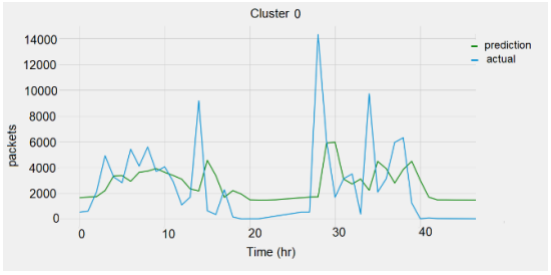
5.2.3 Performance Based on MLP

To run the MLP regressor, the following hyper parameters should be defined [40]:

- **Hidden Layers Sizes**: defines the number of the neurons in each hidden layer
- **Hidden Layers and Output Layer Activation Functions:** defines the activation functions used for the hidden layers and for the output layer
- **Optimizer**: defines the optimization technique
- **Alpha**: defines the learning rate

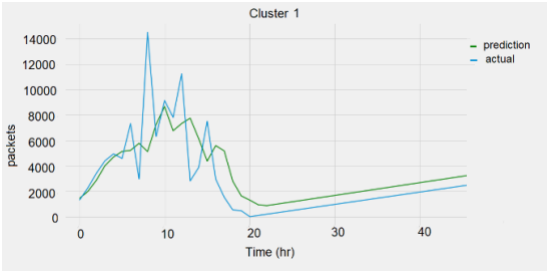
The grid search included three types of Hidden Layer Activation Functions, relu, logistic, tanh or identity function. And for the hidden layers sizes, (30,30,30), (40,40,40) and (50,50,50) were used. Then, for the alpha the search was on 0.1, 0.01, and 0.5. The Output Layer Activation Function was set to be linear activation function in all models and the optimizer used was Adam optimizer which calculates adaptive learning rates for each weigh individually based on both the mean and the variance of the gradients for the weights [77] [37].

In Figure 5.6, the results of a sample base station of each cluster is presented. Under each plot, the hyperparameters value that gave the least RMSE are presented.



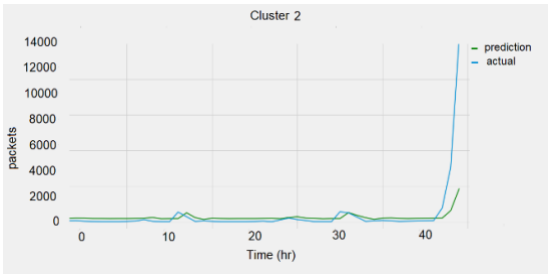
(a)

- Hidden Layers Sizes: (30,30,30)
- Hidden Layers Activation Function: relu
- Alpha: 0.1
- RMSE: 2902



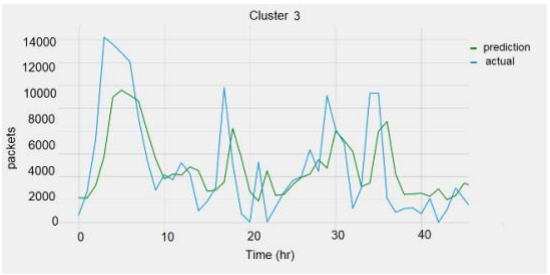
(b)

- Hidden Layers Sizes: (50,50,50)
- Hidden Layer Activation Function: relu
- Alpha: 0.01
- RMSE: 2009



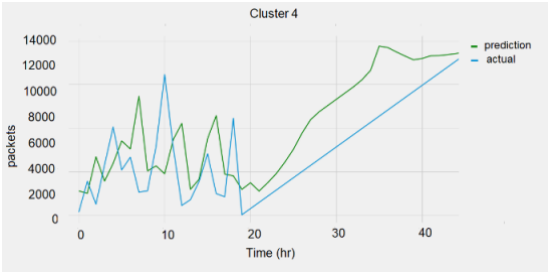
(c)

- Hidden Layers Sizes: (30,30,30)
- Hidden Layer Activation Function:
- relu
- Alpha: 0.01
- RMSE: 1265



(d)

- Hidden Layers Sizes: (40,40,40)
- Hidden Layer Activation Function: relu
- Alpha: 0.01
- RMSE: 1708



(e)

- Hidden Layers Sizes: (30,30,30)
- Hidden Layer Activation Function: relu
- Alpha: 0.1
- RMSE: 795

Figure 5.6: MLP Results

MLP provided better results than SVM, but the results were worse than the SARIMA results, except in the third and fifth cluster. As in the SVM model, MLP did not provide satisfactory results. This indicated that a more advanced architecture such as RNNs may perform better in similar time forecasting problems.

5.2.4 Performance Based on Decision Trees

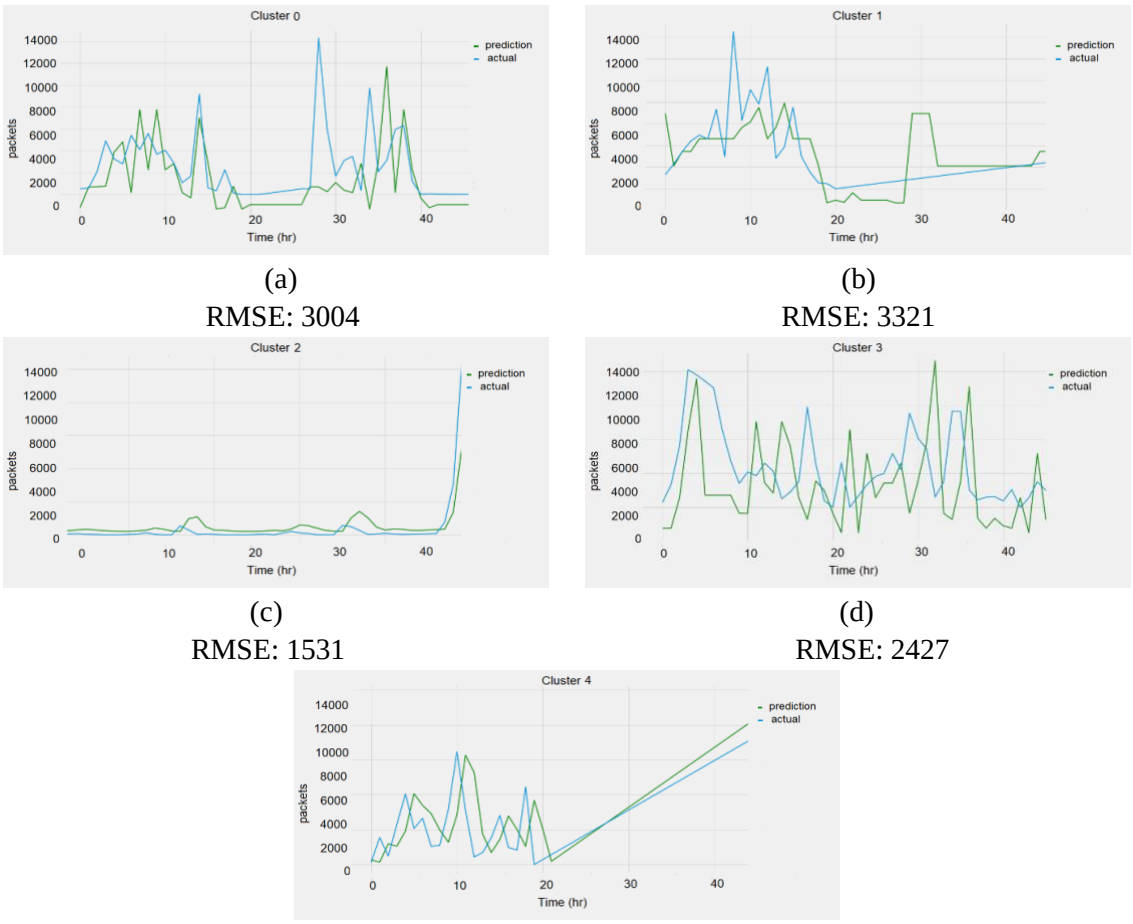
For the Decision Trees Regressor, the following were the most important hyperparameters to define before running the model: [41]

- **Criterion:** defines the split evaluation error function
- **Splitter:** defines the technique of choosing the split on each iteration
- **Max Depth:** defines the tree's depth

The grid search included the following two types of criteria:

- **entropy** for Information Gain
- **gini** for Gini impurity

These two criteria are used interchangeably, however, based on their implementations, gini is less computationally intensive and hence all the criteria of the models were set to gini. The splitting can be determined based on the best result or can be done randomly. The parameter was set to best for all models. For the depth, the "none" value was chosen for all presented models which means that there is no restriction on the depth of the trees. Figure 5. 8 shows the results of a sample base station of each cluster along with the resulting RMSE.



(a)
RMSE: 3004

(b)
RMSE: 3321

(c)
RMSE: 1531

(d)
RMSE: 2427

(e)
RMSE: 1635

Figure 5.7 : Decision Trees Results

Decision Trees provided a higher RMSE in most clusters than the MLP and the SVM. This can be due to the poor behavior of the decision trees algorithm in case of regression problem. As mentioned in Section 2.2, the decision trees were mainly designed for classification problems and do not provide as desirable results in regression problems.

5.2.5 Performance Based on Random Forests

The Random Forests Regressor has some similar hyperparameters as the Decision Trees, as the criterion and three max depth, but the criterion function in Random Forests are either mean squared error or mean absolute error. Also, another important parameter that should be defined is the Number of Estimators which defines how many trees are in the model [78].

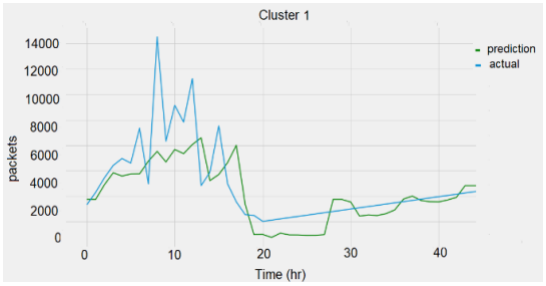
The grid search included different number of hyperparameters in range of 10-25. For the criterion function, Mean Absolute Error (MAE) was selected for all models. MAE is the measure of the average of the absolute errors it can be calculated as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_i - x| \tag{5.1}$$

where N is the total errors number, x is the prediction value and x_i is the true value.

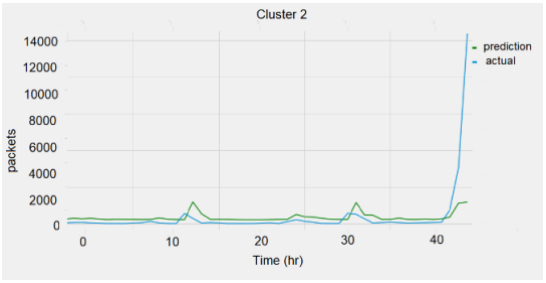
(a)

- Number of Estimators: 24
- RMSE: 3000



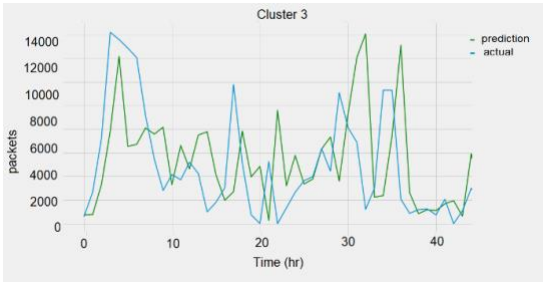
(b)

- Number of Estimators:15
- RMSE: 2172



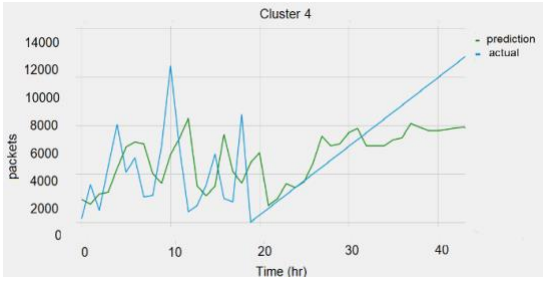
(c)

- Number of Estimators: 10
- RMSE: 1379



(d)

- Number of Estimators: 14
- RMSE: 2464



(e)

- Number of Estimators: 12
- RMSE: 1023

Figure 5.8: Random Forests Results

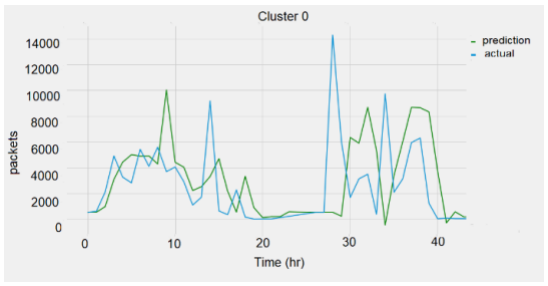
The RMSE produced by the Random Trees algorithm was lower than the RMSE produced by the Decision Trees algorithm. This shows the robust performance of the Random Forests in case of similar regression problems, which may be due to the generalization power provided by Random Forests.

5.2.6 Performance Based on XGBoost

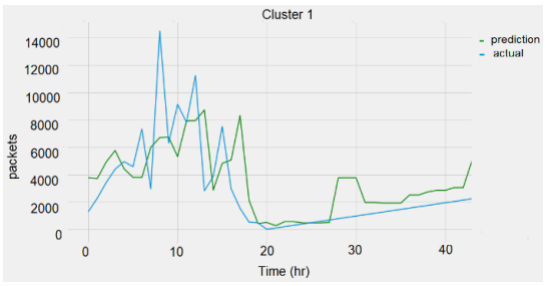
To run the XGBoost Regressor, the following hyperparameters are defined [79]:

- **Booster:** the boosting technique used in the model
- **Gamma:** defines the minimum accepted loss for splitting the tree.
- **Tree Method:** The construction technique chosen to build the trees in the model
- **Max Depth:** defines the tree's depth

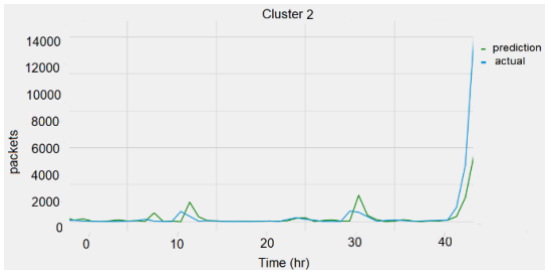
In choosing the hyperparameters, the gamma was chosen to be "0" for all models. And "approx" choice was used for the tree method which uses greedy algorithm to determine the best choice. Then search was on the gbtree, gblinear and dart boosters but gbtree outperformed the other boosters in all models.



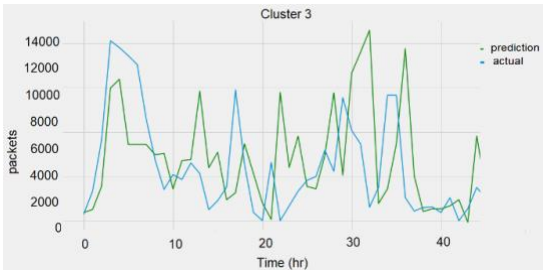
(a)
RMSE: 3520



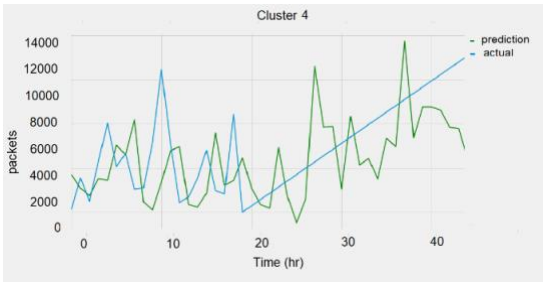
(b)
RMSE: 2416



(c)
RMSE: 1147



(d)
RMSE:2605



(e)
RMSE: 1314

Figure 5.9: XGBoost Results

Unexpectedly, XGBoost provided poor performance when compared to other algorithms, whereas the RMSE of the XGBoost regressor was higher than the Random Forests results for most clusters.

5.3 Performance Based on Deep Learning

Here, the results produced using the RNNs. For both LSTMs and GRUs, the following parameters should be defined [80]:

- **Input dimensions**: defines the input size to the network, for these models it was considered the number of the timeseries in the training data of each cluster

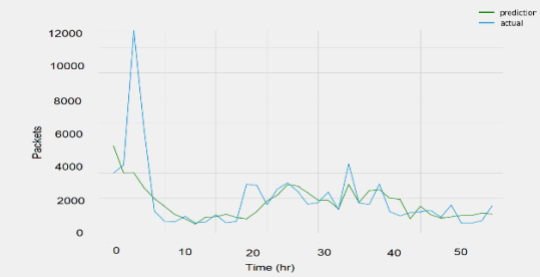
- **Output dimension**: defines the size of the network output, it also considered the number of the timeseries in each cluster because the resulted prediction is for all inputs.
- **Batch Size**: defines the amount of data trained in each iteration in our case this was "32" for all of the networks.
- **Loss function:** the root mean squared error was used to measure the loss of the networks.
- **Optimizer**: "Adam" optimizer was used as the optimizer in all networks.
- **Epochs number**: defines the number of iterations which the model is going to run in our case, 50 epochs were used for all created networks

For each cluster one model was created and trained. 70% of the cluster's data were considered for training and 30% for testing. A sample of base stations belonging to each cluster was considered in the following results and their traffic loads were shown against the forecasting. Then the RMSE, which was calculated as the average RMSE for all base stations belonging to the cluster, was presented.

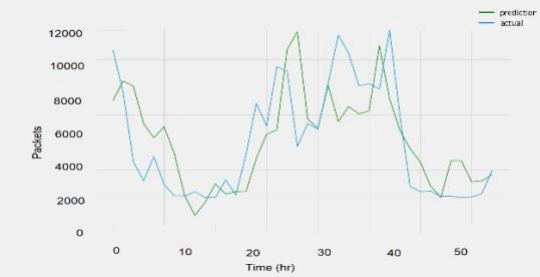
Regarding the architecture of the networks, the LSTM networks included two LSTM layers, a dropout layer to prevent overfitting and a fully connected network layer. The same architecture for the GRU networks was used, except GRUs networks layers instead of the LSTM layers.

5.3.1 Performance Based on LSTM

In the following figures the results from the LSTM networks for several sample base stations of each cluster are presented. Followed by the RMSE results from the model of each cluster. Figures 5.10, to 5.14 show the RMSE of the first to the fifth clusters, respectively. Both the predicted and actual load of two sample base stations in each cluster are plotted.



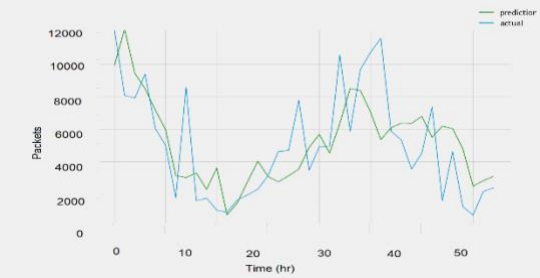
(a)



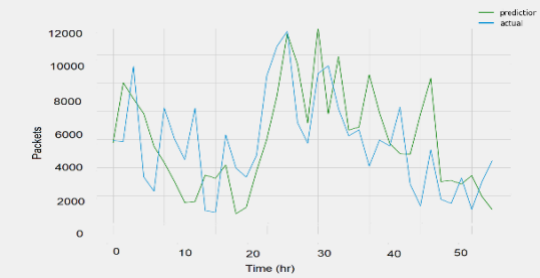
(b)

RMSE: 1532

Figure 5.10: LSTM-Cluster0 Results



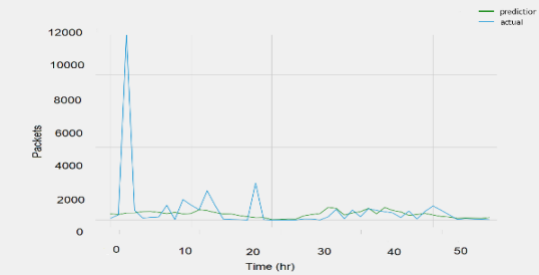
(a)



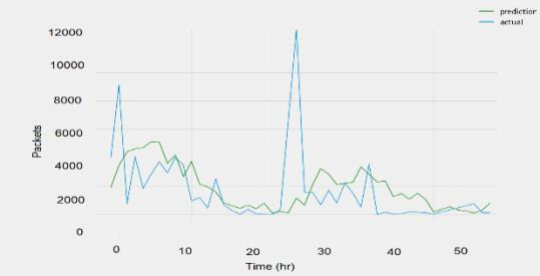
(b)

RMSE: 1140

Figure 5.11: LSTM-Cluster1 Results



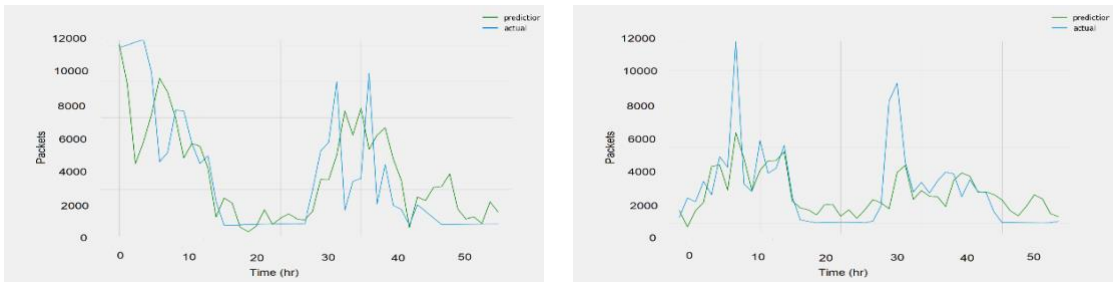
(a)



(b)

RMSE: 690

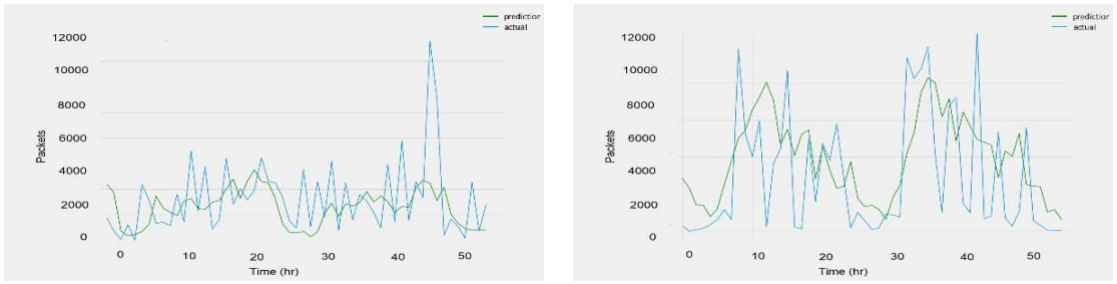
Figure 5.12: LSTM-Cluster2 Results



(a) (b)

RMSE:1091

Figure 5.13: LSTM-Cluster3 Results



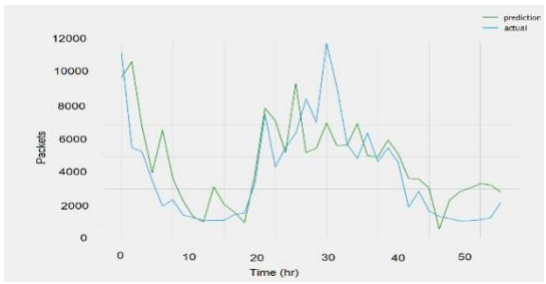
(a) (b)

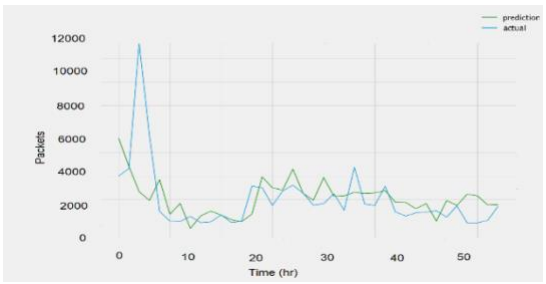
RMSE: 1311

Figure 5.14: LSTM-Cluster4 Results

5.3.2 Performance Based on GRU

The following figures present the results from the GRU networks for a few sample base stations of each cluster. Below each figure of a cluster the RMSE results from the model of each cluster is mentioned. Figures 5.15 to 5.19 show the RMSE of the first to the fifth clusters, respectively. Both the predicted and actual load of two sample base stations in each cluster are plotted.

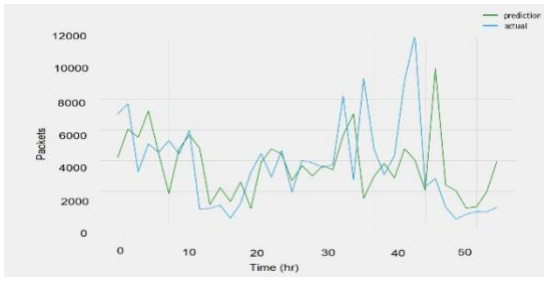


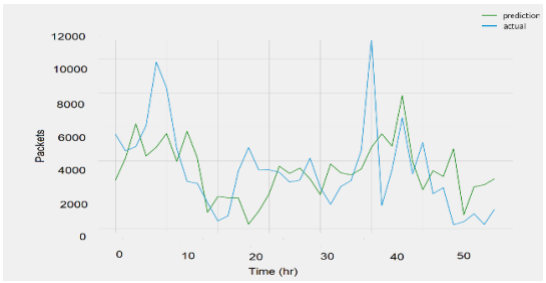


(a) (b)

RMSE: 1205

Figure 5.15: GRU-Cluster0 Results

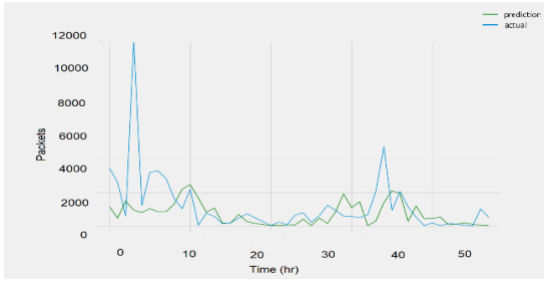


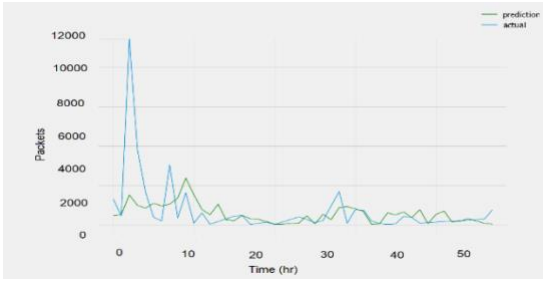


(a) (b)

RMSE: 982

Figure 5.16: GRU-Cluster1 Results

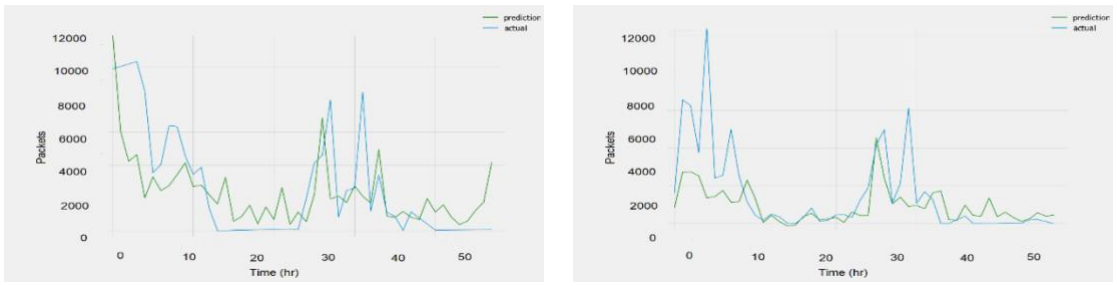




(a) (b)

RMSE: 451

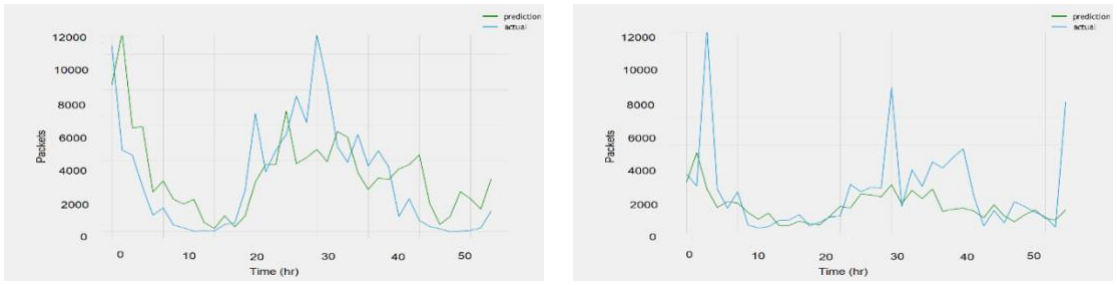
Figure 5.17: GRU-Cluster2 Results



(a) (b)

RMSE: 724

Figure 5.18: GRU-Cluster3 Results



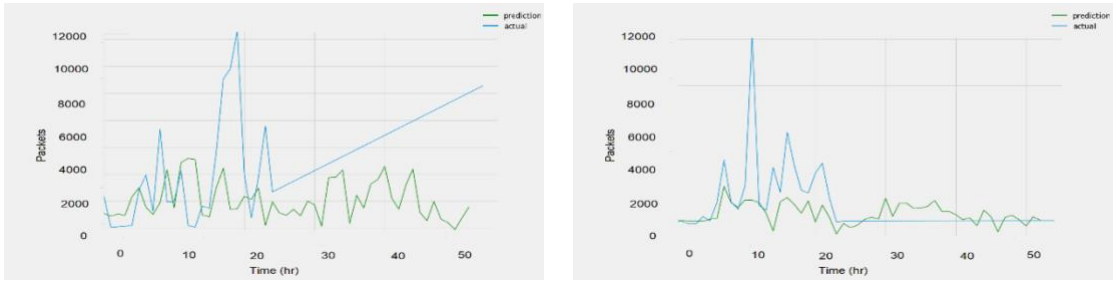
(a) (b)

RMSE: 1113

Figure 5.19: GRU-Cluster4 Results

5.3.3 Performance Based on Un-clustered data

To examine the effect of clustering on the forecasting, the network ran on the complete data. The dataset was split into 70% for training, and 30% for testing. Then the forecasting was tested on them using the same techniques that were used on the clustered version of the data. Figure 5.21 shows the best results produced by LSTM and GRU networks.



(a) (b)

RMSE: 1562

Figure 5.20: Results without clustering

5.4 Online Tools

Recently, online tools were released to perform the time series forecasting process. In this research, two online tools were explored to examine their performance in forecasting the traffic load on the base stations and because of lack of previous works considering our dataset their results were used in this research as a benchmark to compare the performance of our model to the online tool versions. The tools that were used are:

- Amazon DeepAR algorithm [81]
- Facebook Prophet Library [82]

5.4.1 AWS DeepAR algorithm

Amazon Web Services (AWS) is one of the subsidiaries of Amazon created for offering powerful cloud services [83]. AWS uses the form of building blocks for these services; each building block can be used for the development of a different kind of application in the cloud [84]. AWS offers services in several domains; Computing, Machine Learning, Messaging, Migration, Database and Storage are some of these domains. One of the salient services in the Machine Learning domain is Amazon SageMaker [85].

Amazon SageMaker is a service that was created to expedite the process of building, training, and deploying Machine Learning models by covering all of the machine learning workflow. SageMaker offers templates for many Machine Learning algorithms that can be used for different problems [85].

DeepAR is one of the algorithms offered by SageMaker. It is a supervised learning algorithm designed for time series forecasting using autoregressive RNNs. The main advantage of DeepAR algorithm is that it makes use of similarity found between related time series data to improve the forecasting performance by learning a global model from the past data of the existing related time

series. DeepAR inventors claim that DeepAR outperforms the traditional forecasting methods in case of the existence of large number of related timeseries. They have also mentioned that because the model learns from related time series, DeepAR can provide future predictions for new time series with no previous records [81].

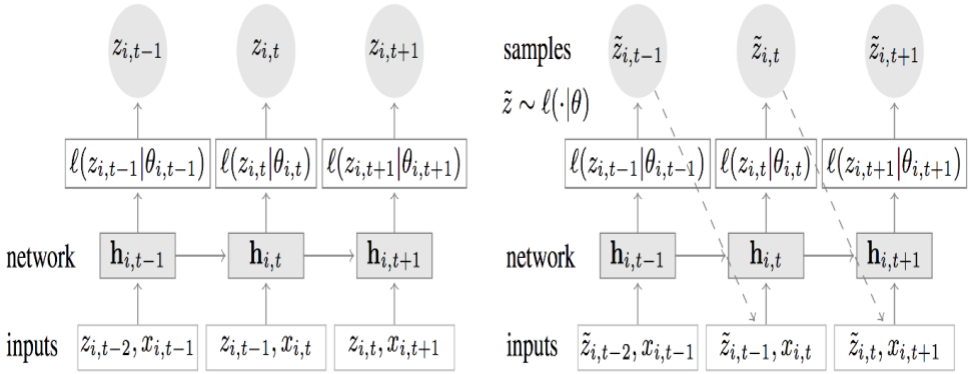


Figure 5.21: DeepAR model Architecture [81]

The DeepAR model uses an autoregressive RNN that comprises the Binomial likelihood to produce probabilistic forecasting [81]. Figure 5.21 describes the architecture of the DeepAR model. The left panel shows the architecture of the network. The network uses the input of the previous observation, (at lag 1) $z_{i,t-1}$, in addition to a set of input variables $x_{i,t}$ along with the previous network output $h_{i,t-1}$, to predict the following time step. The network applies a likelihood function (Gaussian or Negative Polynomial) for training the model. Through the training process, the parametrization used to the likelihood is used to calculate the error. During the backpropagation process, the weights are tuned to change the parametrization parameters until converging to optimal values of the likelihood. The diagram on the right shows that after training the parameters of the network, forward propagation is used to obtain the distribution parameters using the input

z_{i-1} and the optional covariates[1]. The prediction at each time step is represented as a distribution instead of a single value. A sample from the output distribution at the certain time epoch is used as an input to the following time epoch [81].

A windowing procedure was used for the training instances generation to guarantee the total coverage of the whole prediction range. Windows that differ in the starting points are selected for each time series. However, the total length T as well as the relative length of the conditioning and prediction ranges are kept fixed for the training examples. The windowing procedure can be explained as follows: for a window of size t=1 and a time series that ranges from 2020/05/01 to 2021/05/01, the created training examples will be equal to 2020/05/01, 2020/05/02, 2020/05/03, and so on. This procedure allows the model to distinguish the information of the absolute time through variables from the relative position of $z_{i,t}$ in the time series [81].

The model can also include additional features. Consequently, the variables $x_{i,t}$ can be either item-dependent or time-dependent [81]. They can provide additional features, that describes additional information about the item or include variables of high correlation with the output, if they lie in the range of prediction suitable to the model. [86].

To be able to run DeepAR algorithm, the following parameters must be determined first [86]:

context_length: defines the length of previous time points data that the model uses to make the prediction

- **epochs:** defines the maximum number of epochs used for training the model
- **learning_rate:** defines the value of the learning rate used in training
- ***learning_rate_decay:*** defines the rate of decreasing the learning rate

[1] Covariates are the variables that affect the main variables, but they are not variables of interest.

- ***likelihood:*** defines the probabilistic likelihood function used in the model
- ***max_learning_rate_decay:*** defines the maximum number that the learning rate can de decreased by
- ***num_averged_models*:** defines the number of models that DeepAR can be averaged to take the pros of
- ***num_cells:*** defines the number of cells in the RNN hidden layers
- ***num_layers:*** defines the number of hidden layer of the RNN

5.4.2 Facebook Prophet Library

Prophet is an open source project released by Facebook in 2017 [82]. Initially Prophet was developed to forecast different data on Facebook. Then, it was made available as open source tool in both Python and R. Prophet is a very powerful forecasting tool. It is fast because it is built in **Stan** [87] which is a probabilistic programming language written in **C++.** It can handle outliers, missing data, and sudden fluctuations in time series [82].

Prophet can examine multiple forecasting techniques with different parameters. Even experienced analysts may not choose the most accurate model for every problem. But Prophet can powerfully try different variations and choose the best between them.

Prophet also provides the option to include the seasonality and irregularities to the model in order to give the user the option to customize the matching of the historical cycles and the fluctuations of the trends. Furthermore, Prophet allows the user to identify the upper and lower limit of the growth curve which enable the addition of any extra or previous information about the growth or drop of the forecast [82].

Prophet can decompose the time series to three main components, trend, seasonality, and holidays. Then it combines them in an equation as:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon t. \quad (5.2)$$

where g(t) represents the trend, s(t) represents the seasonality, h(t) represent the holidays and εt represents the error or any unusual change by the model [82].

Prophet is built around the idea of Generalized Additive Models(GAM), statistical models that are used in case of nonlinearity in the data by replacing the linear relationship between the output and the regressor with nonlinear smooth function, where in Prophet only the time is used as a regressor, and tries to fit linear and linear functions of time as components [82]. One of GAM's strong points, is that it fits and decomposes quickly if a new source of seasonality or irregularity was added. Hence, the model parameters can be modified interactively [82]. Finally, Prophet design the forecasting problem as a curve-fitting exercise which adds more flexibility when compared to other time series models that explore only the temporal structure in the time series [82].

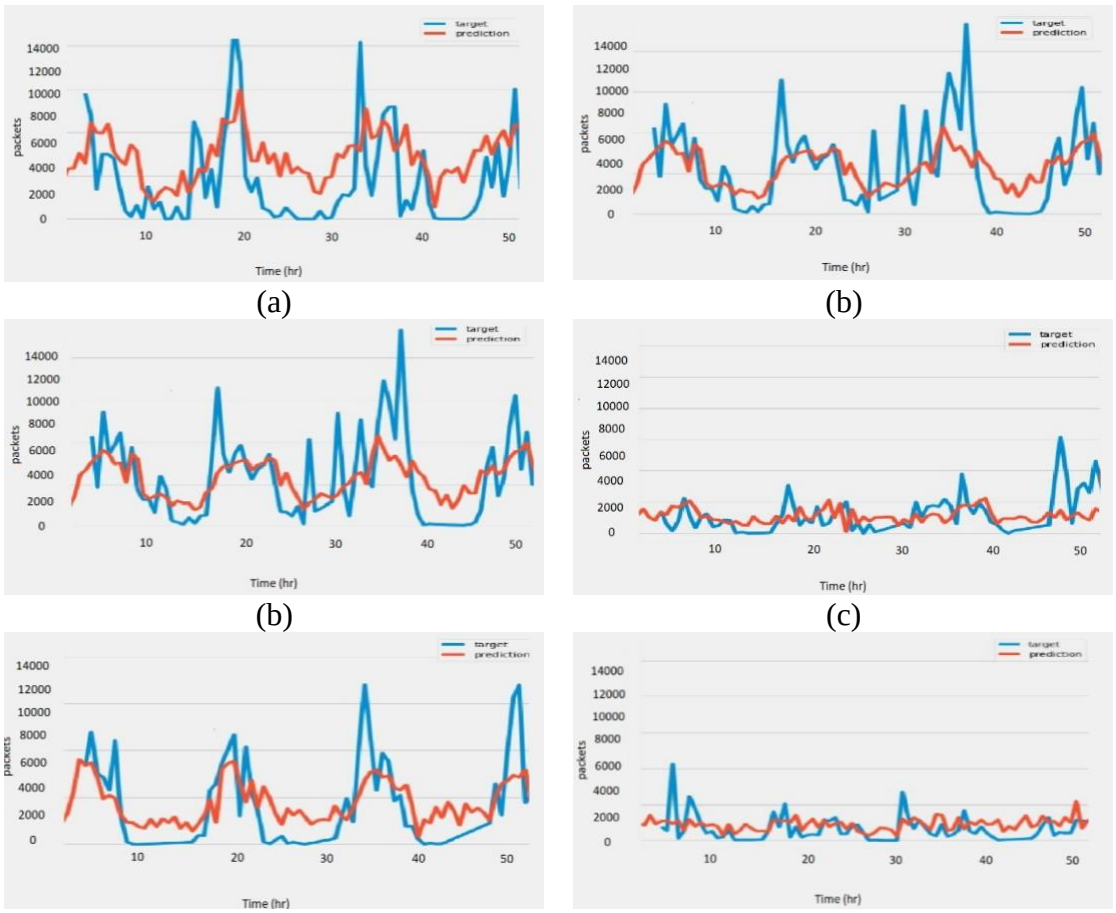
As mentioned above, Prophet reaches the best model by exploring various methods and finding the model with best results. Hence, the process of evaluation of the performance of the forecasts is automated with limited human involvement.

Performance Based on AWS DeepAR Algorithm

DeepAR requires the data to be in a specific JSON format and uploaded on SageMaker. Data preprocessing was performed to be able to use the model. The following hyperparameters were defined to train the DeepAR mode [86]l:

-**Time frequency**: hourly - **Number of cells**: 50 - **Number of Layers**:3 - **Likelihood**: gaussian - **Epochs**: 100 - **Batch size**: 32 - **Learning rate**: 0.0001 - **Dropout rate**: 0.05 - **Loss**: RMSE

Figure 5.22 presents the RMSE score resulted from the DeepAR model along with the plots for the actual load and the prediction of the model for six sample base stations from the dataset.

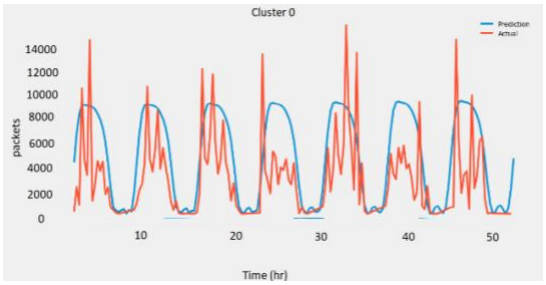


RMSE:1406

Figure 5.22: DeepAR Results

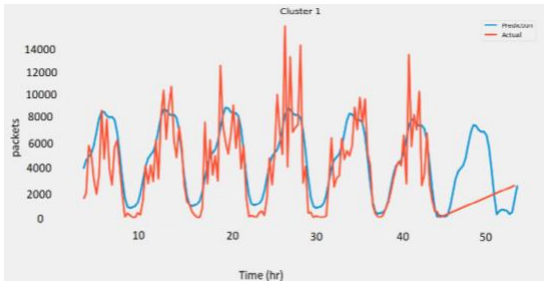
Performance Based on Facebook Prophet Tool

Prophet requires the data to be in a definite format. Hence, a data preparation step was applied. Then each cluster was fed to a separate Prophet model and the following figure, Figure 5.23, shows the RMSE results from each cluster and a plot for the prediction and the real load of a sample base station.



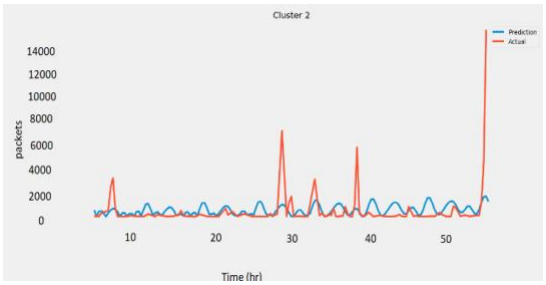
(a)

RMSE: 1419



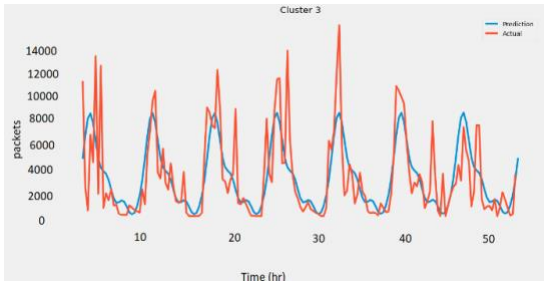
(b)

RMSE: 1263



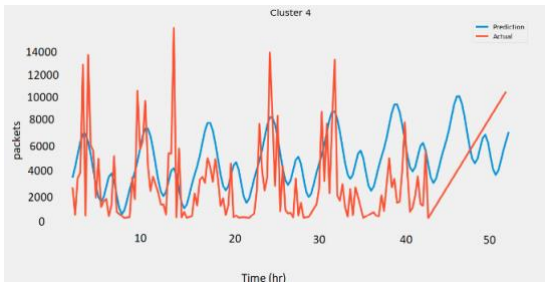
(c)

RMSE: 1253



(d)

RMSE: 1174



(e)

RMSE: 1426

Figure 5.23: Prophet Results

The results provided by the online tools in Figures 5.22 and 5.23 indicate that for this dataset, even the most powerful existing forecasting tools are not going to result in very accurate forecasts. The high variation in the data can be the main cause for this underfitting. Although complex architectures should be able to handle high variation in the data, they require a large amount of data for training. As we mentioned in the limitations of the selected dataset in Chapter 3, the data collection duration was short which resulted in small amount of traffic data provided for each base station. However, these results were

still beneficial to us, as they are indicative of the best performance that can be reached using the provided dataset. Accordingly, we took these results into consideration when we evaluated the examined models.

5.5 Discussion

Figure 5.24 demonstrates the RMSE percentages of all the examined algorithms. It is shown that GRUs provided the lowest RMSE% and Decision Trees resulted in the highest RMSE%. The three bars at the right display the RMSE percentages of the algorithms that were used as references, which are the RNNs on the unclustered version of the data, DeepAR and Prophet.

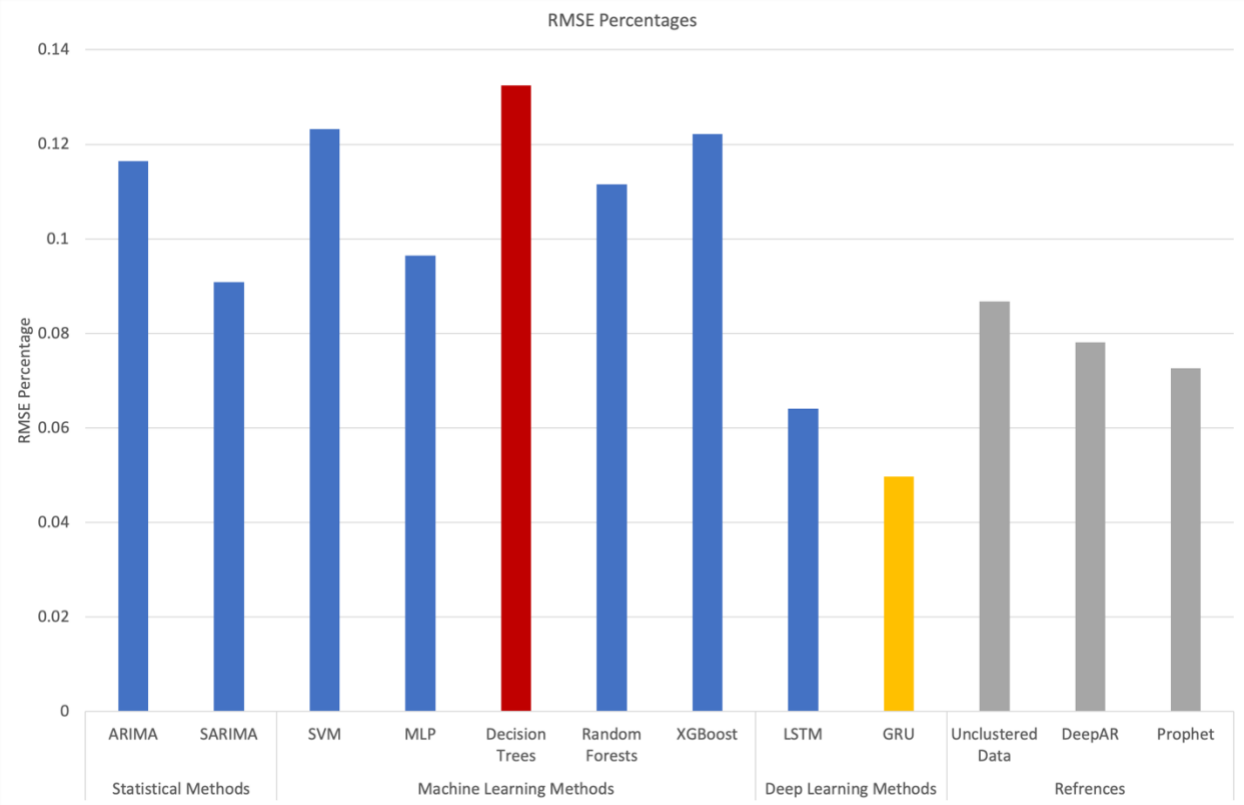


Figure 5. 24: RMSE percentages of all algorithms

The presented results show that the best performance produced was by the RNNs. There results provided by the GRU networks and the LSTM networks were close to each other with a slightly decrease in the RMSE in case of the GRU networks. The error resulting using the clustered version of the data was less than the error produced in case of not clustering. This can show that the network performs better when the similarity of behavior among the training data increases.

Compared to the available online tools, the DeepAR and Prophet, the error produced from our created models was close to the error produced using them.

Regarding Machine Learning and Statistical models, their errors were higher than the error produced using the RNNs. The Statistical Methods' error was less than the error produced using the Machine Learning algorithms, confirming the claim that statistical methods are better than Machine Learning in case of small data [88].

Among the different Machine Learning algorithms that were explored, the MLP algorithm resulted in the lowest error in most clusters. The highest error was produced by the Decision Trees algorithm for most clusters. The poor performance of the decision trees supports earlier belief that Decision Trees are not as good in regression as they are in classification [89]. Although the best performance among all used techniques was achieved by the RNNs, the traditional ANNs using MLP was not as satisfactory as the LSTM and GRU networks. That shows the impact of the memory part in the neural networks on handling the time series forecasting task.

For the ARIMA and SARIMA models, the performance of the SARIMA model is better than the ARIMA, and that is most likely because of the seasonality that is found in the data.

To conclude the following observations can be made:

1. The RNN algorithm performs best in cases where the data includes multiple time series. This is because future records in the time series data depend on the past records and the RNNs architecture includes a memory part that is designed to use the previous observations to predict the next ones.
2. Clustering can significantly improve the forecasting performance because it increases the consistency of the training data

3. The XGBoost algorithm does not always provide better performance than other Machine Learning models. The reason could be that the selected dataset provides relatively few features. The XGBoost algorithm combines multiple decision trees to improve the performance, where each decision tree handles a different subset of the features. In case of a small number of features, limited variations of the features can be made to create the boosted trees. Accordingly, the XGBoost algorithm does not show a significant improvement in the performance compared to the traditional decision trees algorithm.
4. In case of a small amount of data, the statistical models may result in better forecasting than the Machine Learning models. This is because statistical methods are designed to find the relationships between the variables while the Machine Learning models are designed to find a generalized pattern that works on most of the data. Hence, by increasing the data size, the performance of the statistical models does not encounter a significant improvement while the Machine Learning models usually show a notable improved performance.

Chapter 6

Conclusions and Future Work

6.1 Summary

In this research, network traffic load prediction on base stations was modeled as a time series forecasting problem. City cellular traffic dataset, which is a public dataset that provides the traffic load statistics for 13k base stations in China, was used for our project.

Clustering according to behavior was proposed as a solution for the non-similar behavior of the different base stations. Different clustering techniques with different distances metrics were explored to reach the techniques and number of clusters that provides the least error. The least error resulted from using Timeseries K-means as the clustering algorithm with Soft-DTW as the distance metric. Then after trying different numbers of clusters, the base stations were clustered into five clusters and were grouped based on the cluster number. This step was done to train and test each group separately.

Statistical methods, Machine Learning and RNNs which are commonly used time series forecasting techniques, were examined on the dataset. The performance of the algorithms was tested by the resulted root mean square error. RNNs showed better performance than both the statistical methods and the machine learning algorithms. As the results provided, using the clustered base stations was better than the results produced when all of the base stations were trained and tested together.

DeepAR and Prophet, which are two online tools built for the timeseries forecasting problem, by Amazon and Facebook respectively, were utilized as a benchmark to compare the tested algorithms against. The findings showed similar error values for the two groups.

6.2 Future Work

As aforementioned, the dataset used in this work has several limitations. It will be important that future work considers applying and testing the techniques used in this work other datasets that overcome these limitations. For instance, if another dataset provides more information about the load of each individual user, future work could examine the effect of this additional information on the performance of the forecasting algorithms. Also, it is desirable for future work to examine the applied techniques on a dataset with smaller granularities as seconds or minutes. Interesting research topics could be derived if satisfactory results were produced from applying the addressed solutions on datasets with smaller granularities. For instance, if the load of the base stations is available every second, the number of applications that can benefit from this information would be more than the applications that benefit from the prediction of the base station each hour. In addition, future work can use the datasets with smaller granularity for better analysis of the behavior of the base stations on a smaller scale.

Future studies should also examine more clustering and time series forecasting techniques than the selected ones in this thesis and compare the performance of them to the techniques used in this work to see if other techniques can provide lower forecasting errors.

Moreover, the choice of the best forecasting model can be tested by more functional techniques rather than the root mean squared error. This can be achieved by trying the model in real world problems. For instance, the predictions can be used in spike detection applications. In addition, the optimal model would be the model that performs best in detecting the spikes.

Another issue that can be investigated in future work is the forecasting horizon. Because Multistep forecasting increases the uncertainty, which leads to an increase in the error, this research focused

on single step forecasting. Therefore, for future work, models could be examined on higher forecasting horizons.

6.3 Concluding Remarks

The main conclusions of this thesis can be summarized in the following remarks:

1. The loads of the base stations are not always dependent on their locations.
2. Clustering the base stations based on their behavior, before using the time series forecasting methods to predict their loads, can significantly improve the accuracy of their loads forecasting.
3. Several distance metrics can be used to measure the similarity between different time series. Our results have shown that Dynamic Time Warping usually performs better than other metrics.
4. Several clustering algorithms can be used for clustering the time series data. Among them, TimeSeriesKMeans usually results in the lowest error.
5. Normalizing the data before clustering improves the clustering accuracy.
6. For smaller datasets, the statistical methods used for time series forecasting can perform better than machine learning algorithms.
7. RNNs, such as LSTMs and GRUs, usually provide better performance than statistical methods and machine learning for the time series forecasting.
8. GRUs result in lower error than LSTMs in the time series forecasting task.
9. Several online tools, such as AWS DeepAR and Facebook Prophet, exist to ease the process of time series forecasting.

References

[1] "Ericsson Mobility Report Q4 2019 Update," Ericsson, Sweden, 2020.

[2] S. Zhao, "Traffic Prediction Based Power Saving in Cellular Networks: A Machine Learning Method," *Association for Computing Machinery*, pp. 1-10, 2017.

[3] X. Chen, Y. Jin, S. Qiang, W. Hu and K. Jiang,, "Analyzing and modeling spatio-temporal dependence of cellular traffic at city scale," in *IEEE International Conference on Communications*, London, 2015.

[4] Dehai Zhang, Linan Liu,Cheng Xie,Bing Yang and Qing Liu, "Citywide Cellular Traffic Prediction Based on a Hybrid Spatiotemporal Network," *Algorithms,* vol. 2, no. 1, p. 20, 2020.

[5] Anton, H. and Rorres, C, Elementary Linear Algebra, John Wiley & Sons, 2013.

[6] Carmelo Cassisi, Placido Montalto, Marco Aliotta, Andrea Cannata and Alfredo Pulvirenti, "Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining," in *Advances in Data Mining Knowledge Discovery and Applications*, 2012, pp. 71-96.

[7] Gold, O. and Sharir, M., "Dynamic time warping and geometric edit distance: Breaking the quadratic barrier," *ACM Transactions on Algorithms (TALG),* vol. 14, pp. 1-17, 2018.

[8] Ratanamahatana, Vit NiennattrakulChotirat Ann, "Inaccuracies of Shape Averaging Method Using Dynamic Time Warping for Time Series Data," in *International Conference on Computational Science*, Berlin, 2007.

[9] Cuturi, Marco ; Blondel, Mathieu, "Soft-DTW: a Differentiable Loss Function for Time-Series," in *The International Conference on Machine Learning*, 2017.

[10] A. Sardá-Espinosa, "Time-Series Clustering in R Using the dtwclust Package," *The R Journal,* vol. 11, pp. 22-43, 2019.

[11] J. Brownlee, Basics of Linear Algebra for Machine Learning, https://www.mobt3ath.com/uplode/book/book-33342.pdf, 2018.

[12] Paparrizos, John, and Luis Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015.

[13] A. Sard´a-Espinosa, "Comparing Time-Series Clustering Algorithms in R Using the dtwclust Package," *The R Journal,* vol. 12, p. 41, 2019.

[14] Rui Xu , D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks,* vol. 16, pp. 645-678, 2005.

[15] S. Sayad, Real time data mining, Self-Help Publishers Cambridge, 2011.

[16] Kiri Wagstaf, Claire Cardie, Seth Rogers, Stefan Schroedl, "Constrained K-means Clustering with Background Knowledge," in *International Conference on Machine Learning*, 2001.

[17] Charrad, M., Ghazzali, N., Boiteau, V. and Niknafs, A., "NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set," *Journal of Statistical Software,* 2012.

[18] Ketchen, Jr, David J.; Shook, Christopher L., "The application of cluster analysis in Strategic Management Research: An analysis and critique," *Strategic Management Journal,* p. 441–458, 1996.

[19] F.Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing,* vol. 21, no. 8, pp. 1313-1325, 1995.

[20] Tan, P. N., Steinbach, M., ; Kumar, "Cluster Analysis: Basic Concepts and Algorithms," *Introduction to Data Mining,* pp. 467-568, 2005.

[21] A. Kassambara, Practical Guide to Cluster Analysis in R, STHDA, 2017.

[22] P. Jain, "Hierarchical clustering Clearly Explained," 5 July 2019. [Online]. Available: https://towardsdatascience.com/https-towardsdatascience-com-hierarchical-clustering-6f3c98c9d0ca.

[23] Vandewiele, Romain Tavenard , Johann Faouzi , Gilles, "tslearn: A machine learning toolkit dedicated to time-series," [Online]. Available: https://github.com/rtavenar/tslearn. [Accessed 2017].

[24] Chakraborty, Sanjay, and N. K. Nagwani, "Analysis and study of incremental k-means clustering algorithm," in *International Conference on High Performance Architecture and Grid Computing},*, 2011.

[25] Gupta, Nidhi, and R. L. Ujjwal, "An Efficient Incremental Clustering Algorithm," *World of Computer Science and Information Technology Journal (WCSIT),* vol. 3, pp. 97-99, 2013.

[26] Likas, G. Tzortzis ;, A., "The global kernel k-means clustering algorithm," 2008 IEEE International Joint Conference on Neural Networks," in *IEEE World Congress on Computational Intelligence*, Hong Kong, 2008.

[27] A. F. Siegel, Practical Business Statistics (Seventh Edition), Academic Press, 2016.

[28] G. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing,* vol. 50, pp. 159-175, 2003.

[29] J. Brownlee, Introduction to Time Series Forecasting With Python, https://machinelearningmastery.com/introduction-to-time-series-forecasting-with-python/, 2019.

[30] F. N. Sävås, Forecast Comparison of Models Based on SARIMA and the Kalman Filter for Inflation, Uppsala, 2013.

[31] Broomhead, D.S. and Lowe, D., Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[32] A. Ng, CS229 Lecture notes, 2000.

[33] T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997.

[34] R. Waser, Nanotechnology: volume 4: information technology II, John Wiley & Sons, 2008.

[35] S. Paul, "What is Perceptron in Machine Learning?," 8 September 2019. [Online].

[36] L.N.Kanal, "Perceptrons," in *International Encyclopedia of the Social & Behavioral Sciences*, ELSEVIER, 2001, pp. 11218-11221.

[37] J. Brownlee, Better Deep Learning, https://machinelearningmastery.com/better-deep-learning/, 2019.

[38] Lehmann, E.L. and Casella, G., Theory of point estimation, Springer Science & Business Media, 2006.

[39] Willmott, C.J. and Matsuura, K., "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate research*, vol. 30, pp. 79-82, 11 December 2005.

[40] Rumelhart, D.E., Hinton, G.E. and Williams, R.J.,, "Learning representations by back-propagating errors," *nature*, vol. 323, pp. 533-536, 1986.

[41] s.-l. developers, "Decision Trees," [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/master/doc/modules/tree.rst.

[42] J. Quinlan, "Induction of decision trees," *Machine learning*, pp. 81-106, 27 Feburary 1986.

[43] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J.,, "Classification and regression trees," *International Group*, vol. 432, pp. 151-166, 1984.

[44] J. Rocca, "Ensemble methods: bagging, boosting and stacking," 22 April 2019. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

[45] I. Reinstein, "XGBoost, a Top Machine Learning Method on Kaggle, Explained," [Online]. Available: https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html.

[46] Chen, T. and Guestrin, C., "Xgboost: A scalable tree boosting system.," *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* , pp. 785-794, 2016.

[47] Yoav Freund and Robert E. Schapire, "A decision-theoretic generalization of on-line learning," *Journal of Computer and System Sciences,* vol. 55, p. 119–139, 1997.

[48] J. Diederich, Artificial neural networks: concept learning, IEEE Press, 1990.

[49] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078,* 2014.

[50] Hochreiter, S. and Schmidhuber, J., "Long short-term memory," *Neural computation,* vol. 9, pp. 1735--1780, 1997.

[51] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 5 Feburary 2014.

[52] H. Abu-Ghazaleh and A. S. Alfa, "Application of Mobility Prediction in Wireless Networks Using Markov Renewal Theory," *IEEE Transactions on Vehicular Technology,* vol. 59, pp. 788-802, 2010.

[53] Aleš Svigelj, Radovan Sernec , Kemal Alic, "Network traffic modeling for load prediction: a user-centric approach," *IEEE Network,* vol. 29, no. 4, pp. 88-96, 2015.

[54] F. Tang, Z. M. Fadlullah, B. Mao and N. Kat, "An Intelligent Traffic Load Prediction-Based Adaptive Channel Assignment Algorithm in SDN-IoT: A Deep Learning Approach," *IEEE Internet of Thing,* vol. 5, no. 6, pp. 5141-5154, 2018.

[55] T. Miyagi, M. Iizuka and M. Morikura, "A novel route reconstruction practically for P-MP communication over wireless ad-hoc network," in *IEEE 51st Vehicular Technology Conference Proceedings*, Tokyo, 2000.

[56] William Su, Sung-Ju Lee, and Mario Gerla, "Mobility prediction and routing in ad hoc wireless," *International Journal Of Network Management,* vol. 11, no. 1, pp. 3-30, 2001.

[57] Miao, Guowang and Song, Guocong, Energy and spectrum efficient wireless network design, Cambridge University Press, 2014.

[58] J. Hu, W. Heng, G. Zhang and C. Meng, "Base station sleeping mechanism based on traffic prediction in heterogeneous networks," in *International Telecommunication Networks and Applications Conference*, Sydney, 2015.

[59] S. E. Hammami, H. Afifi, M. Marot and V. Gauthier, "Network planning tool based on network classification and load prediction," in *IEEE Wireless Communications and Networking Conference*, Doha, 2016.

[60] Zhang, Rongpeng Li Zhifeng Zhao Xuan Zhou Honggang, "Energy savings scheme in radio access networks via compressive sensing-based traffic load prediction," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 4, pp. 468-478, 2012.

[61] M. Alzenad, A. El-Keyi, F. Lagum and H. Yanikomeroglu, "3-D Placement of an Unmanned Aerial Vehicle Base Station (UAV-BS) for Energy-Efficient Maximal Coverage," *IEEE Wireless Communications Letters*, vol. 6, no. 4, pp. 434-437, 2017.

[62] Li, Rongpeng, Zhifeng Zhao, Xuan Zhou, and Honggang Zhang, "Energy savings scheme in radio access networks via compressive sensing-based traffic load prediction," *Transactions on emerging telecommunications technologies*, vol. 25, pp. 468-478, 2014.

[63] Dong Xin, Wentao Fan, Jun Gu, "Predicting LTE Throughput Using Traffic Time Series," *ZTE Communications*, vol. 4, p. 014, 2015.

[64] Iandola, F., Moskewicz, M., Karayev, S. Girshick, R., Darrell, T. and Keutzer, K., "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[65] M. Larson, "Descriptive statistics and graphical displays," *Circulation*, vol. 114, pp. 76-81, 2006.

[66] Friendly, M. and Denis, D., "The early origins and development of the scatterplot," *Journal of the History of the Behavioral Sciences*, vol. 41, pp. 103-130, 4 September 2005.

[67] L. Kazmier, Schaum's outline of theory and problems of business statistics, McGraw-Hill New York, 1976.

[68] G. Van Brummelen, Heavenly mathematics: The forgotten art of spherical trigonometry, Princeton University Press, 2012.

[69] T. W. Liao, "Clustering of time series data—a survey," *Pattern Recognition*, vol. 38, no. 11, pp. 1857-1894, 2005.

[70] Saeed Aghabozorgi, Teh Ying Wah, Tutut Herawan, Hamid A. Jalab,Mohammad Amin Shaygan, and Alireza Jalali, "A Hybrid Algorithm for Clustering of Time Series Data Based on Affinity Search Technique," *The Scientfic World Journal*, 2014.

[71] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. Pérez and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, pp. 243-256, 2013.

[72] Kim, Minho; Ramakrishna, R. S., "New indices for cluster validity assessment," *Pattern Recognition,* vol. 26, no. 15, pp. 2353-2363, 2005.

[73] T. Motzkin, "The euclidean algorithm," *Bulletin of the American Mathematical Society,* vol. 55, pp. 1142-1146, 6 January 1949.

[74] P. Senin, "Dynamic Time Warping Algorithm Review," *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA,* vol. 855, p. 40, 17 October 2009.

[75] S. Palachy, "Detecting stationarity in time series data," 21 July 2019. [Online]. Available: https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e638.

[76] scikitlearn_developers, "sklearn.svm.SVC," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[77] Kingma, D.P. and Ba, J., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[78] scikitlearn_developers, "sklearn.ensemble.RandomForestRegressor¶," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[79] xgboost_developers, "XGBoost Documentation¶," [Online]. Available: https://xgboost.readthedocs.io/en/latest/.

[80] tensorflow-developers, "tf.keras.layers.LSTM," [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

[81] Salinas, David ; Flunkert, Valentin ; Gasthaus, Jan ; Januschowski, Tim, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting,* 2019.

[82] Taylor, Sean J ; Letham, Benjamin, "Forecasting at scale," *The American Statistician,* vol. 72, pp. 37-45, 2018.

[83] I. Amazon Web Services, "Cloud computing with AWS," [Online]. Available: https://aws.amazon.com/what-is-aws/.

[84] Y. Pant, "An Overview of AWS Storage — Part 1," 23 May 2019. [Online]. Available: https://blog.cdw.com/data-center/an-overview-of-aws-storage-part-1.

[85] I. Amazon Web Services, "Amazon SageMaker," [Online]. Available: https://aws.amazon.com/sagemaker/.

[86] I. Amazon Web Services, "DeepAR Hyperparameters," [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/deepar_hyperparameters.html.

[87] "Stan," [Online]. Available: https://mc-stan.org/.

[88] Bzdok, D., Altman, N. & Krzywinski, M, "Points of significance: statistics versus machine learning," *Nature Methods,* vol. 15, pp. 233-234, 2018.

[89] L. Li, "Classification and Regression Analysis with Decision Trees," 15 May 2019. [Online]. Available: https://towardsdatascience.com/https-medium-com-lorrli-classification-and-regression-analysis-with-decision-trees-c43cdbc58054.