Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

Traffic prediction optimization with clustering

Authors

Péter Regőczi

Henrik Berényi

Supervisors

Dr. Vilmos Simon

Norman Bereczki

Contents

Kivonat 1

Abstract 2

1 Introduction 3

2 Related work 5

3 Methodology 9

3.1 Unsupervised machine learning 9

3.2 Clustering 9

3.2.1 DBSCAN and HDBSCAN 9

3.2.2 Distance-metrics 11

3.3 Traffic prediction using machine learning 12

3.3.1 SVR 12

3.3.2 LSTM 13

4 Creating the simulation environment 17

4.1 Introducing the SUMO framework 17

4.2 Creating the simulation network 18

4.3 Generating traffic and anomalies 19

5 Data preprocessing 21

5.1 Overview of the preprocessing pipeline 21

5.2 Map matching 22

5.3 Parameter calculation and aggregation 23

6 Training the machine learning models 26

6.1 Clustering the data 26

6.1.1 Adjusting the parameters of the algorithms 27

6.1.2 Visualizing the results 28

6.2 Training prediction models 30

6.2.1 SVR 31

6.2.2 LSTM 32

7 Results 35

7.1 Evaluate without clustering 36

7.2 Evaluate clustering 40

8 Conclusion ... 43
Acknowledgements ... 44
Bibliography ... 45

// Kivonat is the abstract

Abstract

The number of vehicles that participate in transportation has been increasing every year which made traffic congestion a larger issue. Traffic congestion has both economic and health implications as the more time a vehicle spends in traffic the more fuel it consumes and the more harmful substances it emits.

The advancements of IoT sensors used in both the infrastructure and the vehicles themselves provide large quantities of accurate data about the state of traffic. Cooperative Intelligent Transport Systems (C-ITS) can make use of this data [1] by utilizing machine learning algorithms to create models that accurately recognize or even predict the presence of a traffic anomaly or congestion [2].

Clustering is an unsupervised machine learning method which divides data points into several groups called clusters based on correlation between their features. Large traffic networks have all kinds of different road types with different behavior. Dividing these road sections and assigning similar ones to common clusters could save computational resources, because it is not necessary to train a model for every road, just for every cluster. By training these models using the clusters separately would also help eliminate bias, as the objective is to predict traffic parameters on specific road types instead of a whole network.

The goal of our work is to predict traffic parameters with machine learning models and use clustering to reduce computational cost. We propose a solution which is capable of grouping road segments with similar parameters, thereby their parameters can be forecasted with the same prediction model. Our method optimizes computational resources and makes the solution suitable for real-time application. We have evaluated the clustering solution's performance with two prediction models.

1 Introduction

Traffic congestion has become a major problem in our world. It has a negative impact on economics, environment, health, and the quality of life. In the U.S., people spend more and more time in congestions, which caused $179 billion cost in 2017, as a result of wasted time and fuel [3]. In 2017 congestion added 88 billion extra hours of travel in the U.S. and 3.3 billion gallons of fuel was consumed [3]. This study [4] shows that most of the individual's health is affected due to traffic congestion, their main symptoms are mental stress, headache, tiredness, unexpected sweating, and breathing difficulty. Traffic congestion can have long term effects, as it is associated with circulatory disease, heart disease, lung cancer, asthma, and acute lower respiratory infections in children [5]. In classrooms, traffic pollution can affect students' attention, concentration, and reaction time, due to nitrogen dioxide and elemental carbon [6]. As the largest portion of greenhouse gasses coming from transportation (29%), traffic has a large impact on the environment [7]. In traffic congestion, vehicles spend more time on the roads, their travel time increases, thereby they emit more CO_2 [8]. Besides these, congestion wastes the time of people, so they can spend less time with their families, friends, and they can be late for work. Roads are noisy during congestion, which can be annoying for pedestrians, and increased travel time costs people more money because of the increased fuel consumption. Papers citated above show that traffic congestion is a major problem in modern cities, and we need solutions that can help optimize traffic flow.

Traffic prediction can help optimizing traffic flow with traffic signal control [9] and it is helpful for navigation apps. Traffic lights with fixed timers are not optimal for traffic flow, as they cannot use the information of the state of the traffic. It is possible to optimize traffic lights with the output of a machine learning based prediction model, so we can manage traffic real time, potentially prevent congestion [9]. For emergency services, it is important to know about the traffic state in real time, so they can reach their destination faster, which can even save lives.

As all road parameters are different, it would be necessary to build a prediction model for every road segment to get good results, which has a large computational cost. The goal of our work is to save computational capacity by grouping road segments into clusters and build a prediction model for every cluster separately. In this work, SUMO (Simulation of Urban MObility) is used for creating traffic simulation with different scenarios, to reproduce real life traffic flow. The output

of the simulation is the vehicles trajectory data, with 10Hz. As the vehicles trajectory data is not completely accurate in real life, map matching should be used for correcting these. Valhalla's Map Matching Service is used for correcting these in datasets, which come from real traffic, but because we used a simulation to generate data, it was not necessary. We aggregated the dataset for every road segment and calculated the traffic flow parameters for 5-minute periods. The output parameters from the aggregation are road length, mean speed, mean travel time, flow, density, and a flag that shows whether a vehicle passed the road in the 5-minute period. We used HDBSAN to group similar road sections by these parameters and creating clusters. With the use of clustering, it is not necessary to build a prediction model for every road segment, just for every cluster, thereby we can save computational cost. We used SVR and LSTM for traffic forecasting. Our method uses data from the previous 25 minutes to predict traffic flow parameters to 10 minutes. The predicted parameters are mean travel time, flow, and the aforementioned flag. We evaluated the results of the models with and without clustering and compared their results.

2 Related work

In recent years, data collection and processing became more efficient due to the developments made in the field of artificial intelligence and IoT sensors. New and more efficient machine learning models have been introduced and made publicly available. Traffic congestion prediction is a commonly researched area with various goals and methods. Some works aim to forecast the presence of congestion using historical data and some of them are even capable of real-time prediction [10].

Priambodo et al. [11] used a data from real world sensors located in Aarhus, Denmark to create their solution. They proposed a machine learning model based on a probabilistic reasoning model, the Hidden Markov Model (HMM). In order to determine congestion, they defined a metric called congestion index. This index is based on speed and occupancy values extracted from the data and represents the level of congestion on a given road segment. They also introduced an unconventional road segment clustering method using Grey Level Co-occurrence Matrix and spectral clustering [12] [13]. As a result of clustering, road segments that have the same amount of congestion at around the same time of day are grouped together. They managed to produce superior results compared to existing solutions using HMM.

Mondal and Rehena [14] used clustering to identify congestion patterns using real time traffic data. They used the K-means clustering method, which is one of the most well-known clustering algorithms. It is widely used and is capable of producing adequate results for most datasets [15]. The algorithm is configured with a target number that defines the number of centroids in the dataset, which represent the centers of the clusters. The centroids are randomly selected at first and then moved at every iteration using the distance values between the data points and the centroid while minimizing the value of a defined loss function. This results in the various data points separated into K number of clusters with the nearest mean (centroid) [16]. They defined four clusters based on density and speed. The model produced good results and was able to accurately determine the predefined clusters. The output of this model could be used to train machine learning models that could produce higher accuracy when trained using the clustered data independently.

Huang et al. [17] introduced a congestion prediction method using the random forest algorithm. They have also used the DBSCAN clustering algorithm to divide the data points based on the level of congestion using velocity and flow as features. DBSCAN is a density-based clustering method

that can determine the amount of clusters by itself and can also recognize noise. The model produced great results, but according to their evaluation, there is room for optimization and more features should be considered such as weather reports or holidays.

Many papers have used machine learning and deep learning techniques for traffic flow prediction, and achieved remarkable results [18]. Lin et al. [19] used combined Support Vector Regression (SVR) and K-Nearest Neighbors Algorithm (KNN) to predict traffic flow. SVR is a commonly used model in regression tasks, which tries to fit a line on the training data, while KNN averages K number of training data, which are the closest to the test data point, to return its output. While training SVR, KNN learned to predict the prediction errors, which reduced root mean squared error significantly compared to the traditional SVR. The paper published by Shen presents [20] a model for long-time traffic speed prediction with XGBoost and considered spatial and temporal dependencies to improve the model. XGBoost is an ensemble of decision trees which combines weak models to create stronger ones.

Long Short-Term Memory (LSTM) is a widely used deep neural network which is designed for handling sequential data, and it uses gates to control the information flow through the network. Mondal and Rehena [21] have showed that stacked multivariate LSTM gives better results than univariate LSTM or Autoregressive Integrated Moving Average models. Gated Recurrent Unit (GRU) is also popular in traffic prediction, it is similar to LSTM, as it also uses gates, but less, thus its complexity is smaller. Bi-directional Gated Recurrent Unit (bi-GRU) is able to process the input in both directions by combining two separate GRUs. Wang et al. [22] compared bi-GRU with single GRU, and they found the two models' errors almost equal, since the difference in mean absolute error is only 0.48% between them. Mean absolute error (MAE) averages the absolute differences between the real and the predicted value. A popular method to forecast traffic flow parameters is hybrid CNN-LSTM, which appears in multiple papers [23] [24] [25]. CNN uses convolutional layers to extract spatial characteristics, while LSTM extracts temporal characteristics. Transformers are more and more popular in deep learning, it uses attention to track relationships in sequential data. Reza et al. [26] used multi-head attention-based transformer to predict traffic flow, compared it with recurrent neural networks and SVR. Their results showed transformer's superiority on both MAE and MSE metrics. Mean squared error (MSE) averages the squared differences between the between the real and the predicted value. Unsupervised models can be used for traffic prediction, stacked autoencoder (SAE) uses dimension reduction to extract

features from the input data. Kailasam et al. [27] proposed deep architecture model with stacked SAE, and trained it with greedy algorithm, which means the network trains layer-by-layer. To fine-tune the model, they used Back Propagation method, which adjust the network's weights and biases, to improve the prediction's accuracy. Djenouri et al. [28] built a pipeline with graph CNN to forecast traffic flow. They reduced the noise and removed outliers from the dataset in the pre-process, then made the prediction with the mentioned model.

In this work, two widely used models have been chosen: SVR and LSTM. The aim was to try both simple and deep machine learning models, which are robust and accurate. LSTM has become one of the most popular prediction models recently and is also widely used in hybrid models, as it is built for sequential data, such as time series [29]. While deep learning techniques are used more frequently in traffic forecast papers, SVR often appears in comparisons. It is specialized for regression problems, such as traffic prediction.

Researching the topic of traffic congestion prediction, we have come across numerous solutions that involve popular machine learning algorithms and models. These models were proven to be quite successful on their own, however they were often trained on a single traffic network raising the question that they might be biased. Training these models also take a considerable amount of time and computational resources, especially if the model relies on neural networks or other types of deep learning algorithms. An advantage of the model presented in this paper is that it provides a solution for scalability by reducing the total amount of time needed to train traffic prediction models.

The motivation of this paper is to create a model that would account for the biases of these models by separating road segments that behave similarly, so that the model would be able to see correlation between the roads but not the whole network at any given time. This would also conserve computing power and time because training would happen only on the clusters and not each individual road. To achieve this, we used an unsupervised machine learning technique called clustering.

Clustering not only enables the presented methodology to train and test the machine learning models faster, but it might also produce more accurate results because of the separation of the outliers. Conserving time and resources are beneficial even with some tradeoffs. If a given model requires a fraction of the time to be trained using clustered input but produces marginally worse

results, it could be considered as a viable solution. Our work aims to preserve the same performance of models used in intelligent transport systems as well as reducing the time needed to set them up.

3 Methodology

3.1 Unsupervised machine learning

Unsupervised machine learning is a learning method that is used to train models using unlabeled data. It is used to reveal the underlying structure of the dataset and group or associate similar data points with one another [30]. Common unsupervised learning techniques consist of association rules, dimension reduction and clustering.

Association rules are used to discover relationships between features in the dataset [31]. It is commonly used for market basket analysis, for example learning the relationship between products in an online store or uncovering the listening habits of users on a music streaming service.

Dimensionality reduction is used to increase the speed of machine learning models as well as transforming a dataset to a form that is easier to visualize. The dimensionality of a given dataset is defined by the number of features it has. Dimensionality reduction reduces these features while also keeping the integrity of the data intact [32]. Popular algorithms include Principal Component Analysis (PCA), Autoencoders, Uniform Manifold Approximation and Projection (UMAP) and t-distributed stochastic neighbor embedding (t-SNE).

3.2 Clustering

Clustering is a technique that divides unlabeled data based on similarities into groups called clusters. There are various clustering algorithms that each present a different approach of defining the clusters. The most commonly used clustering algorithms are centroid-based like the k-means algorithm, density-based like DBSCAN or hierarchical like HDBSCAN [33].

3.2.1 DBSCAN and HDBSCAN

Density based clustering algorithms offer a different approach by working on the assumption that clusters are dense regions in space, separated by regions of lower density. This eliminates the problem of outliers as these methods are able to identify them and leave them out of any cluster. The most popular density-based clustering algorithm is Density-Based Spatial Clustering of Applications with Noise (DBSCAN). DBSCAN is efficient and easier to configure than k-means.

It is able to determine the total number of clusters by itself and does not need to be told beforehand. DBSCAN only requies two parameters: epsilon and minPoints. Epsilon is the radius of the circle in which points are considered to be neighbors of one another [34]. MinPoints defines the least number of neighboring points required to form a core point [34]. The algorithm works by establishing circles (or hypherspheres in higher dimensions) around the data points with the radius of the given epsilon and classifies them as core point, border point and noise (see *Figure 1*). A core point is a data point whose boundary contains at least minPoint amount of other data points. If the neighboring points are less than minPoints then it is classified as a border point. If there are no neighbors, then it is classified as noise [34].

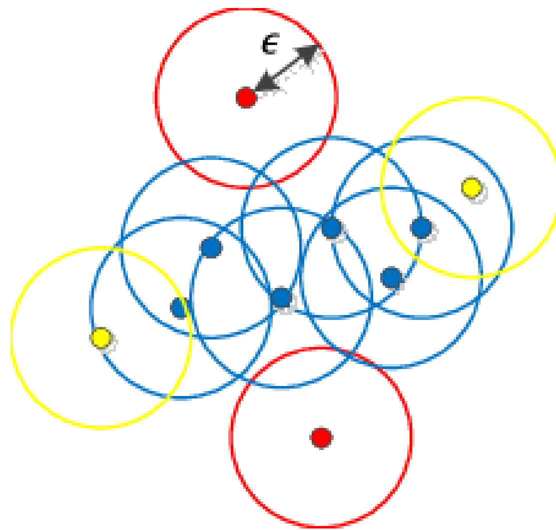

Figure 1: DBSCAN's data point classification [35]

HDBSCAN is a hierarchical extension of the DBSCAN algorithm. It is more efficient as the flat clustering approach by utilizing many density thresholds. This makes the algorithm computationally less expensive and more accurate as well [36].

The algorithm does not require an epsilon parameter as the previously introduced flat counterpart, because it redefines how the distance measurement is implemented. The new distance metric is the mutual reachability distance which keeps dense points close to each other but pushes sparser points away. During execution, a minimum spanning tree and cluster hierarchy is built, from which the clusters are extracted.

HDBSCAN focuses more on higher density clusters. This is also emphasized by its most important parameter, the minimum cluster size which defines the least amount of data point required to be close to each other for it to be considered a cluster [36].

In conclusion, HDBSCAN proposes a significant improvement over DBSCAN and is able to identify clusters of any size or shape while also being less resource intensive and more accurate [36].

3.2.2 Distance-metrics

Distance between data points can be measured in different ways which has a considerable amount of effect on the outcome of the algorithm. In Euclidean geometry, the Euclidean distance is the shortest path between any two given data points. The city block or Manhattan distance is measured by taking the sum of distances between the x and y coordinates (see *Figure 2*). There also exists a distance measure for time series, called Dynamic Time Warping (DTW), which compares the points of two time series by mapping the corresponding ones to one another. This creates a more efficient distance measure than the Euclidean distance. The input data of the clustering process is provided in the form of a multivariate time series, therefore we have chosen DTW as our distance matrix calculation method.

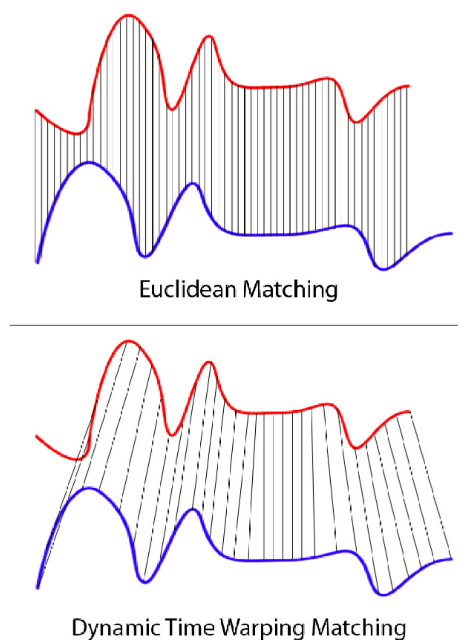

Figure 2: visual representation of commonly used distance measures [37]

3.3 Traffic prediction using machine learning

3.3.1 SVR

SVR is a supervised machine learning algorithm, and is a modification of SVM (Support Vector Machine) but used for regression problems. It predicts the value of the target variable based on the input features by fitting a line on the training dataset. Unlike Linear Regression, SVR can work well on nonlinear data with the usage of the kernel trick.

The goal of the SVR is to fit a hyperplane on the training data which minimizes the prediction error, and keeps the model's complexity low. In SVR, there is a tube, called Ɛ-insensitive and every data point's error will be disregarded which falls inside this tube (see *Figure 3*). The points outside the tube are training errors, SVR tries to minimize their losses. The closest data points to the hyperplane inside the tube are called Support Vectors, which are a small number of the training vectors, and these will give the regression line. By getting rid of most of the training vectors, and keeping just the Support Vectors, SVR reduces its computational load, since the final model becomes smaller.

The goal in the training is to reduce the deviation of the points outside the tube, while keeping the model complexity low. The more data points are inside the margin, the better.

$$min \frac{1}{2} ||w||^2 + C \sum_{i=1}^{n} |\xi_i|$$

$$Constraints: \quad |y_i - w_i x_i| \leq \varepsilon + |\xi_i| \tag{3.1}$$

The main formula for SVR is shown in *3.1* equation. SVR's main hyperparameters are ε, C and kernel. Parameter ε stands for the maximum error, so SVR only calculates with errors that are larger than ε. Smaller ε means narrower margin, while bigger ε means wider margin which is more error tolerant. C is the tradeoff between empirical risk and regularization terms [38]. If the C value is small, SVR tolerates errors more, the complexity of the model is low, but it may not train well, empirical risk can be high. On the other hand, if parameter C is high, it punishes errors more, which will fit the training data well, but it can cause overfitting, and high model complexity.

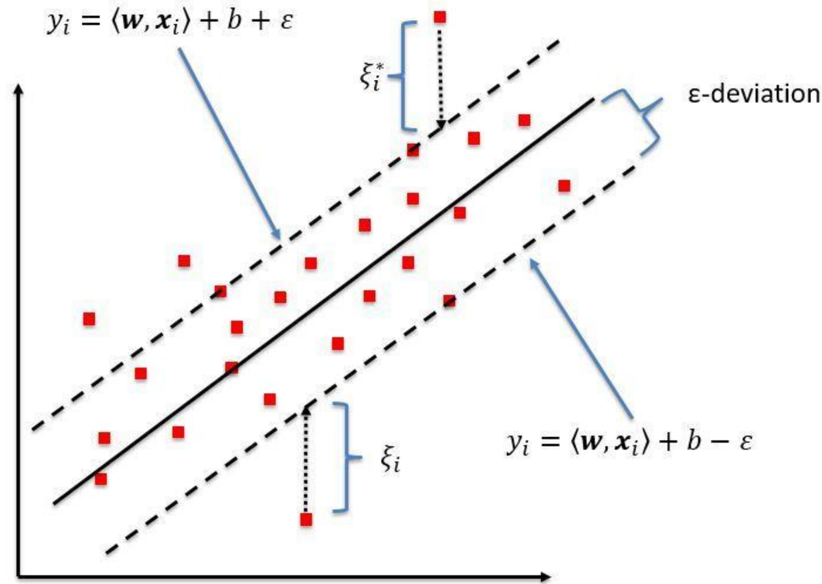

Figure 3: Illustration of SVR [39]

SVR can manage non-linear datasets with the usage of the kernel trick. Kernel function adds a new dimension to the input vector with non-linear transformation, so the system can be treated as linear. The four main SVR kernels are Linear, Radial Basis Function (RBF), Polynomial and Sigmoid. Overall, we need to choose these parameters appropriately to get the best results with SVR.

SVR is not preferable on large datasets, as its complexity is quadratic, so a ten times bigger dataset means a hundred times more operations. In this case, Linear SVR or other regression models can be a better choice.

3.3.2 LSTM

LSTM is a type of Recurrent Neural Network (RNN), which is capable of learning long-term dependencies. Since LSTM uses data from the previous steps, it is a suitable choice for working with serial data, time series, such as traffic flow.

RNN is designed for serial data, as it has a recurrent loop, which allows the model to use the previous step's information, besides the current input. This solution allows the information to persist, thus giving memory to the model.

Neural networks are trying to minimize their loss function during training, so in every iteration it adjusts its parameters to get better results. To find out how to adjust the parameters, neural networks are using backpropagation, which means that we compute the gradients of the loss function with the respect of weights and biases. During backpropagation we are moving backwards layer-by-layer and compute the gradients by the chain rule. Therein lies RNN's disadvantage, which is the vanishing gradient problem [40]. Because RNN has recurrent loops, we need to backpropagate through time, and as we go deeper, there can be a lot of gradient multiplication according to the chain rule. If the gradients are small, we multiply many small numbers which gives us a result of almost zero. This is called vanishing gradient, which prevents us from training the model well. Also, if the gradient values are big, the result will explode, which is called exploding gradient. We calculate the new weights and biases by multiplying the gradient and the learning rate, but if the gradient is too small, the change is almost zero, so our model does not learn. This shows that RNN cannot learn long term dependencies.

Against the vanishing gradient problem, LSTM uses gates to learn long term dependencies. These gates control what is the relevant information from the input, what the network should remember from the past, and generate the output.

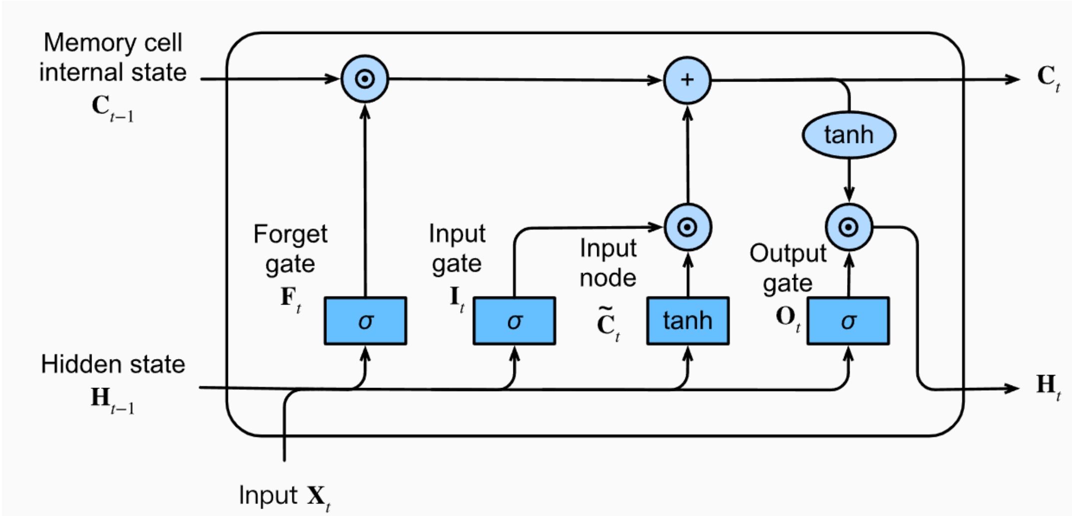

Figure 4: LSTM structure [41]

The structure of the LSTM network is shown in *Figure 4*. The core of the LSTM is the cell state, which lets the information persist. Gates can add or remove information from the cell state, hence it controls the flow of the information.

The forget gate controls which information is relevant from the past, and which should be removed. It uses the previous hidden state and the current input, then a sigmoid activation function compresses the values between 0 and 1. Zero means that the information can be removed completely, while numbers close to one are the most important. We multiply the sigmoid function's output with the cell state, which updates the memory cell (*Equation 3.2*).

$$f_t = \sigma\left(W_f * [h_{t-1} * x_t] + b_f\right) \tag{3.2}$$

The input gate and input node decide what information of the input should be added to the cell state. The input gate creates weights for the input vector with a sigmoid function, decides which value to update, while the input node creates a vector from the candidate values. We multiply them and add them to the cell state (*Equation 3.3*).

$$i_t = \sigma(W_i * [h_{t-1} * x_t] + b_i)$$
$$\widetilde{C_t} = \tanh(W_C * [h_{t-1} * x_t] + b_C) \tag{3.3}$$

With the outputs of the forget gate, the input gate, and the input node, we have updated the cell state (*Equation 3.4*):

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{3.4}$$

The last part of the LSTM unit is the output gate, which determines what is important from the cell state to make the prediction. To get the output, which is also the next hidden state, we push the cell state into a tanh function, which returns a value between -1 and 1, then multiply it with the output of the sigmoid function, so we get just the important values (*Equation 3.5*).

$$o_t = \sigma(W_o * [h_{t-1} * x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t) \tag{3.5}$$

Overall, LSTM can learn long term dependencies, which makes it a robust, and widely used model in the field of deep learning, however its disadvantage is complexity, and big amount of training data.

4 Creating the simulation environment

Although the amount of traffic data being collected increased steadily over the past few years, it is still difficult to obtain them both because they are expensive and might not contain all the parameters that we would like to use to train out models [42]. By simulating traffic scenarios, we have control over the whole network, its roads and even the traffic anomalies that potentially lead to congestions. It is also easier to create large amounts of data using a simulation, which is especially useful when training neural networks. This chapter introduces the simulation framework, describes the created simulations and shows how data can be obtained from them.

4.1 Introducing the SUMO framework

Simulation of Urban MObility (SUMO) is an open-source simulation framework used to create versatile traffic simulations. It is developed by the German Aerospace Center (DLR) and licensed by the Eclipse Foundation. SUMO is not just a single software, it offers a wide variety of tools that assist in the creation, evaluation, and visualization of the simulations [43] [44].

The simulation software and its capabilities can be extended through its public API called Traffic Control Interface (TraCI). The TraCI library is available in multiple programming languages, with varying numbers of features [45]. There are some third-party simulation softwares that make use of this API, most notably Artery, which is a V2X simulation software using multiple network simulation frameworks to implement ETSI ITS-G5 protocols. Artery uses SUMO as a backend through TraCI and simulates the wireless communication between vehicles and infrastructure separately [46].

Traffic networks can be created using the “netedit” application, which allows the user to define the road segments, lanes and their connections, junctions, traffic lights and additional points of interests. There is also a tool which can import a network from OpenStreetMap (OSM). Networks are treated as the base of any simulation and therefore do not contain any traffic information themselves. This makes networks reusable in multiple scenarios that might use a different number or types of vehicles.

After a network has been designed, the ongoing traffic should be defined separately to create an executable simulation. Traffic configuration consists of three main elements: routes, trips, and vehicles.

Routes can be defined by listing all the edges in the order a vehicle should travel through them. A vehicle always enters the simulation on the first edge and exits the simulation as soon as it reaches the last edge. Routes can be external or internal based on where they are defined. External routes can be associated to any vehicle while internal ones are declared within a vehicle block and therefore are associated to a single vehicle or vehicle type.

Vehicles can be instantiated individually with an assigned route or generated using the flow block. Flows generate vehicles using the defined distribution, parameters, and route. It is easier to simulate vehicles that would take the same route because a single flow can handle any number of vehicles.

SUMO is also capable of creating the route of a vehicle by itself with the use of trips. Trips only require two parameters, a departing and a destination edge. After defining a trip, the software calculates the fastest route the vehicle will take. This behavior can be fine-tuned with parameters like “viaJunction” that will include designated junctions in the route.

To create a complete simulation, a “sumofile” should be created which references the files where the network, routes and trips are defined. Simulations can be executed in the SUMO GUI app by importing the created sumofile. However, apart from debugging the simulation, the GUI app can be difficult to handle in complex cases. To generate the necessary outputs in the desired format, the more versatile command line should be used. The CLI version of SUMO has various options for creating outputs: it is possible to generate aggregated and disaggregated vehicle and edge information, and with the use of TraCI, any information about the simulation can be queried at any time. Output files are generated in XML format and could be converted to a CSV for consumption.

4.2 Creating the simulation network

Serving as the base, networks are the most important part of any simulation. We made sure that our network was chosen carefully and was able to replicate real-life scenarios. Therefore, we have decided to use the OSMWebWizard tool to import the base of the network from a real city [47]. Using the tool, we selected certain roads from the city of Budapest. There are 58 edges in total, each representing a single lane of a road. Every road has two lanes, going in each direction. There

are two junctions with traffic lights. *Figure 5* shows the visualization of the network. Each lane is assigned a unique identifier.

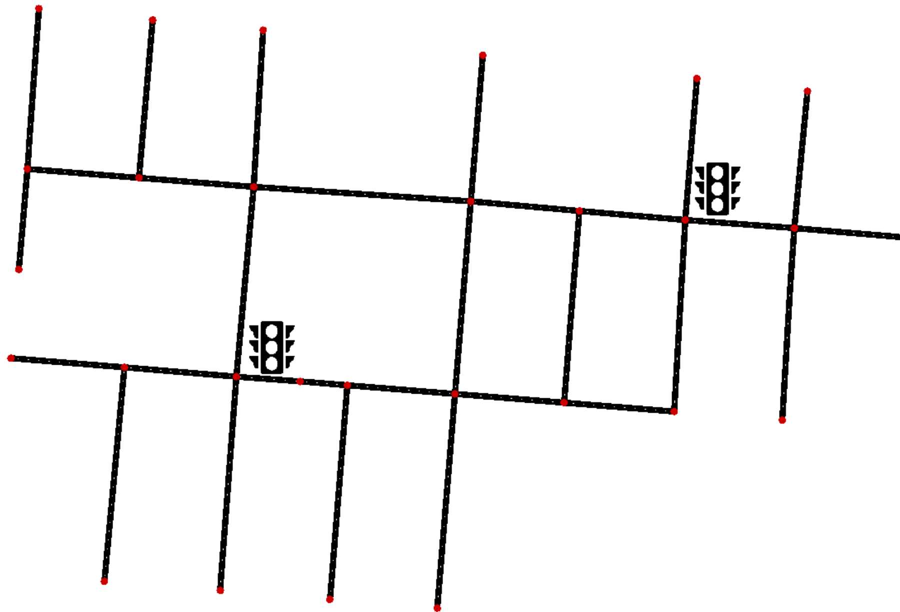

Figure 5: the network of the simulation

4.3 Generating traffic and anomalies

Many things must be considered when creating traffic for a simulation. The traffic needs to be tailored to the network, so that it conforms to what real life traffic would look like.

Although SUMO can handle custom vehicle types with varying sizes and driver behaviors and even public transportation such as buses, we have decided to focus solely on cars to keep the task less complex and potentially achieve better results. Therefore, we defined multiple types of cars with some of them having abnormal behavior, causing congestion. There are "slow cars" which have a lower maximum speed and cars that stop either by performing an emergency brake or by gradually slowing down and stopping at the side of the road.

Regular traffic is generated using the OSMWebwizard tool mentioned in Chapter 4.1. This tool not only imports maps but is also capable of generating traffic. It will leave an executable script with editable parameters. We have edited these parameters, more specifically the maximum speed and density parameters to create the base of the traffic. The previously mentioned anomalies are then injected using "flow" blocks. Each flow has a predefined route and vehicle type associated

with it, as well as a parameter that tells the software how many of these vehicles should be generated per hour.

In total, we have created six scenarios, each representing a real-world single day scenario:

- Morning peak
- Morning peak with anomalies
- Traffic under unfavorable weather conditions
- Traffic under unfavorable weather conditions with anomalies
- Weekend peak
- Weekend peak with anomalies

Peaks have higher maximum speed and density while there are fewer cars driving slower during certain weather conditions.

The output of the simulation is focused on the data of individual vehicles. We used the netconvert tool to simulate accurate coordinates and generated Floating Car Data (FCD) output to obtain vehicle speed, position and trajectory.

5 Data preprocessing

Since we get the raw output of the SUMO simulation, we need to preprocess it to get the necessary parameters. The raw output from the simulation are the vehicle's id, speed, position, angle, and type. We used the map of the network, the vehicle's id and its trajectory data to aggregate the dataset, thus creating a data frame which contains the parameters for every road in every five minutes. These parameters are road length, mean speed, mean travel time, flow and density, and a flag that shows whether any vehicle passed the road in the current timeframe. Both the clustering and prediction model use these or some of these parameters, so it is important to calculate their values accurately. The preprocess code was written in Jupyter notebook, for managing data frames we used pandas and NumPy libraries [48] [49] [50].

5.1 Overview of the preprocessing pipeline

The output of the SUMO simulation gives vehicles parameters such as id, speed, position, angle and type, and saves it in a csv file. We read the csv file into a pandas DataFrame, which is a 2-dimensional labeled data structure, which has rows and columns. Then, we selected the necessary columns, which are: vehicle id, x coordinate, y coordinate. If the trajectory data is not accurate, we need to use map matching, which corrects these coordinates. This occurs when we use real time data. Trajectory data can get noisy, it can cause problems in the preprocess later on. Because we worked with simulated data, our trajectory data was accurate enough, therefore we did not use map matching while testing. However, because of the inaccuracy of real-life datasets, map matching usually cannot be omitted, thus we built our system to be able to use map matching, which will be presented in the next subsection.

Trajectory data does not give us information about which road the vehicle is on. To overcome this problem, we fitted polygons onto the map to identify the roads, and to calculate traffic flow parameters for each road. If we place a polygon in all junctions, then every pair of polygons defines a single road section. We iterated through every vehicle's data to match them with the proper road section and calculated its travel time. The vehicle's travel time on a single road section is calculated from the time difference between entering the entry and exit polygon. We created a data frame which contains every vehicle's travel time on each road. Finally, we calculated the aggregated parameters from the mentioned data frame, which are mean travel time, mean speed, flow, density,

flag that shows whether a vehicle passed the road in the current timeframe, and the length of the road, which is equal to the distance between the polygons.

5.2 Map matching

While working with real life datasets, trajectory data can be inaccurate because of the noise. For example, in CAM messages longitudinal position error's median can be 3 m [51]. This error could cause problems in our solution. While calculating travel times, we detect the timesteps when the vehicle is inside a polygon. If our trajectory data is inaccurate, and a vehicle enters an intersection, coordinates can show wrongly that the vehicle is off the road, so it is possible that we do not detect the vehicle's entry if the polygon is not big enough. We could place a bigger polygon on the junction, but it would increase the travel time calculation's error. So, the best solution is map matching, which corrects these data, thus the coordinates will be on the road.

For map matching, Valhalla's Map Matching Service [52] has been chosen, which is an open-source routing engine and uses OpenStreetMap data. It has multiple modules but the relevant for map matching is Meili. It matches the sequence of coordinates to the road network using Hidden Markov Model and Viterbi Algorithm. Since Valhalla has a docker image, the easiest way to use it is through docker. After pulling the docker image, and downloading the OpenStreetMap data of the country, we can turn on the server for Valhalla.

We use a python function [53], which gets pandas DataFrame with the vehicle's trajectory data as input and gives back the same DataFrame with the corrected longitude and latitude coordinates. This function first converts the DataFrame into a json file and formats it. It makes the request for the docker server, reads the respond, then converts it back to pandas DataFrame.

As we mentioned before in *Chapter 5.1*, map matching has not been used in the evaluation, because it was not necessary, as the data comes from an accurate simulation. But while working with inaccurate real-life data, map matching is an essential part of the preprocessing pipeline.

5.3 Parameter calculation and aggregation

The goal of the aggregation is to get every road parameter in every five minutes from the input dataset. These parameters are *road length*, *mean speed*, *mean travel time*, *flow, density* and a *flag*, which is True if no vehicle passed to road in the 5-minute timeframe. Travel time is the length of time to pass through the road, and mean travel time is the average of travel times in the timeframe. The mean speed of a single vehicle is calculated by dividing the road length with the travel time of the vehicle. By averaging the mean speeds of every vehicle in the current timeframe, the required mean speed parameter is resulted. Flow shows how many vehicles passed the road in one hour, while density is the flow divided by road length. The core idea in aggregation is to place a polygon in every intersection, thus a pair of polygons identify a road section, and we can detect when a car enters and when it leaves the road. For minimizing the travel time calculation's error, we need to place a polygon as small as possible, which just covers the intersection. However, we need to consider how frequently the vehicle sends data, because if it is possible that it enters and leaves an intersection between two samplings, then we might miss the detection. In our simulation vehicles send data in every 0.1s, which is the minimum generation time of CAM message [54], and the speed limit is 50km/h, which means that it moves 1.39 meters between two samplings if it obeys the speed limit. Since our intersections have width about 6 meters it cannot cause problem.

For placing the polygons in the intersections, we used Mapbox's geojson.io [55], where we can draw polygons in the map manually, and it gives back their coordinates. These coordinates are stored in a python file, we give ID to each polygon, and put them in a NumPy array. The coordinates are converted into Shapely's [56] Polygon geometry type because it has some beneficial functions, such as giving back the center of the polygon, or determining whether a point is inside the polygon or not. As we mentioned before (*Chapter 5.1*), two polygons identify a road section, so for defining roads, we created a NumPy array, which contains its ID, the entry polygon's ID, the exit polygon ID, and the road length. Road length is calculated by the distance between the centers of the entry and exit polygons. For this calculation we used GeoPy's great_distance function, which computes the shortest distance between two points on earth's surface [57].

The dataset contains each vehicle's id, longitude and latitude in every 0.1 seconds. To use Shapely's functions, we converted each vehicle's longitude and latitude to Shapely's Point

geometry type. Next, we calculated all vehicle's travel times on every road that it used. For this, we iterated through each vehicle's every data point and detected when it entered a polygon. First, the vehicle enters the network, soon gets to an intersection and gets detected using the polygon method described above. Since this is the first time it is inside a polygon, we cannot calculate any parameter, so only the timestep and the polygon's ID that it entered are stored. The vehicle moves on and when it enters a polygon again, the system detects it and compares the current timestep with the previously stored timestep. The difference between the stored entering and leaving time returns the vehicle's travel time on the road, while the stored and current polygon define the road section. We replace the stored polygon ID and timestep with the current ones and repeat the process until the vehicle leaves the road. Using this method on all vehicles gives a dataset which contains all the calculated travel times on every road. To optimize the computation of this process, it is not necessary in every timestep to determine if the vehicle is inside a polygon . If the shortest road's length, the vehicle's max speed and the data sending frequency are known, some data points can be skipped. For example, in our simulation the shortest road's length is 98 meters, the max speed is 60km/h and SUMO's data sending frequency is 10Hz. This means that a vehicle can move 1.67 meters between two samplings, thus on the shortest road it sends at least 55 data points while moving from the entry polygon to the exit polygon. If we skip the next 50 data points after the vehicle leaves the entry polygon, because we know that it takes 55 steps to reach the next intersection, then we save computation time, and speed up the process.

Since the data sending frequency is 10Hz, it causes errors in the travel time calculation. When a vehicle enters or leaves a polygon, we do not know when it exactly entered in the 0.1 second interval. Thus, it causes a maximum of 0.1 second error on both the entry and the exit polygon, which is 0.2 second summary. The formula of the relative error is shown in *Equation 5.1*, where f is the data sending frequency, s is the road length, and v is the mean speed of the vehicle. In our simulation, roads have an average length of around 150 meters, and the speed limit is 50 km/h, which gives 1.85% maximum relative error. This error does not affect our method significantly.

$$\frac{\Delta t}{T} = \frac{2 * \frac{1}{f}}{\frac{s}{v}} = \frac{2 * v}{s * f} \tag{5.1}$$

The last step is to create the aggregated data frame and calculate its parameters from the previously mentioned data frame that contains the travel times. The aggregated data frame should contain every road's parameters in 5 minute frames. We iterate through each timeframe in every road and select those vehicles' data which were on the road in the current timeframe. This data contains the vehicle's ID, the timestep when it left the road and its travel time. These values and the length of the road are enough to calculate the aggregated data frame parameters. For mean travel time, we simply average the selected vehicles' travel times. For mean speed parameter, we calculate the mean speed of each vehicle by dividing road length with travel time, then averaging them. Flow shows how many vehicles passed the road in an hour, thus we count the number of vehicles in the 5-minute frame, and multiply it by 12 to stretch the time to one hour. Density is calculated by dividing the flow with the road length. It is possible that no vehicle passes the road in a timeframe, thus there is no travel time data. This can happen in a low-density road when the road is empty, or in a really crowded network, where vehicles cannot enter the intersection for a long time. In this situation, we calculate the parameters according to the speed limit, so the mean speed will be the speed limit, and the mean travel time is the road length divided by the speed limit. Also, we created a flag, which shows if no vehicle passed the road in the current timeframe.

Overall, the aggregated data frame contains the traffic flow parameters of each road in five minutes long timeframes. Aggregation allows us to use clustering and to train predictions models, therefore it is essential in our solution.

6 Training the machine learning models

After it has gone through the preprocessing pipeline, the aggregated data is then forwarded to clustering. The algorithm focuses on time series clustering and produces an output that contains the list of road segments per cluster as well as the outliers. The number of clusters is determined automatically. Using this information, SVR and LSTM machine learning models are then trained on each cluster and outliers separately.

6.1 Clustering the data

The input of the clustering pipeline is the concatenation of the preprocessed and aggregated data from every simulation. It contains the road length, flag, mean speed, mean travel time, flow and density as features in the form of multivariate time series.

The input data cannot be used with a clustering algorithm and must go through some processing steps first. A popular solution for time series clustering is dynamic time warping (DTW) [58]. We have used DTW to calculate the distance matrix that can then be used as an input for the clustering algorithm.

After the distance matrix has been constructed, it could be used to perform clustering. However, we have decided to implement dimension reduction (DR) as an intermediary step. Dimension reduction has proven to be useful as it allows for visualization and therefore better understanding of the dataset. Among the many solutions, Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) has proven to be the most accurate [59]. UMAP has a property that allows for setting the distance metric. If precomputed is chosen, the required input is a distance matrix. At this step, we set the target dimension and provide the precalculated distance matrix as the data input. Plotting the data points already makes it apparent which cluster certain road segments belong to and can reveal connections between them.

The last step is the clustering itself using HDBSCAN from the scikit-learn python library. The output from the previous step is passed to the algorithm to produce the clusters.

6.1.1 Adjusting the parameters of the algorithms

Clustering relies just as heavily on its parameters as it does on the input data itself. Since we use three different steps using three different libraries, we must ensure that each of them is parameterized so that it produces desirable results. The method implements dynamic time warping has a quadratic time complexity, and therefore has parameters that limit its total runtime. Fortunately, the dataset and the runtime are not large enough to warrant the use of any of these parameters. UMAP has multiple parameters that revolve around the relations of the datapoints. The most important one is n_neighbors, which determines whether the process should prioritize local or global structure. In our case, focusing on local structure has produced better results and we have decided on a value of 15. *Figure 6* presents a comparison between a small and large n_neighbors value.

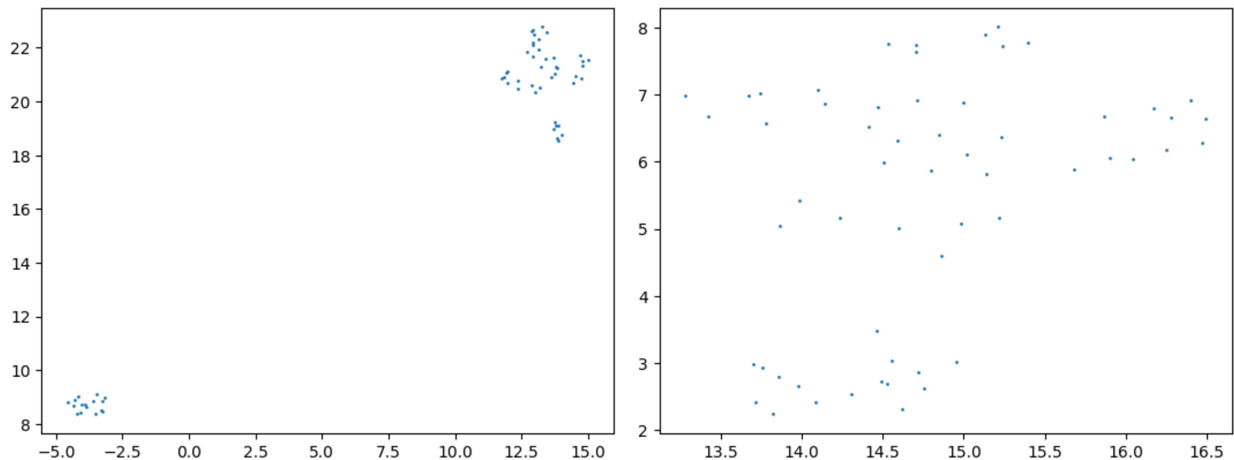

Figure 6: UMAP outputs using a small (left) and high (right) n_neighbors value

The metric parameter of UMAP is set to “precomputed” since the precalculated distance matrix is provided by the methodology described in Chapter 6.1. DBSCAN’s most important parameter, the epsilon does not exist in the case of HDBSCAN but that does not mean that we have no ability to control cluster structure. Minimum and maximum cluster sizes, as well as the cluster selection method can be specified. The two types of selection methods are Excess of Mass (eom) and leaf. We have decided on leaf as it produces more accurate clusters [60].

6.1.2 Visualizing the results

Visualization of machine learning models is just as important as constructing them. It helps us to understand the dataset and compare the results of using different parameters. One of the reasons why we decided to use dimension reduction is to be able to plot the data points on a two-dimensional chart.

The result of the clustering algorithm can be observed using a scatter plot. The individual data points are represented by dots and colored according to the cluster that they belong to (see *Figure 7*). Noise is determined as defined in Chapter 3.2.1 and classified as outliers. They are displayed as crosses to indicate that they do not belong to any cluster. Additionally, the recorded centroids can also be plotted.

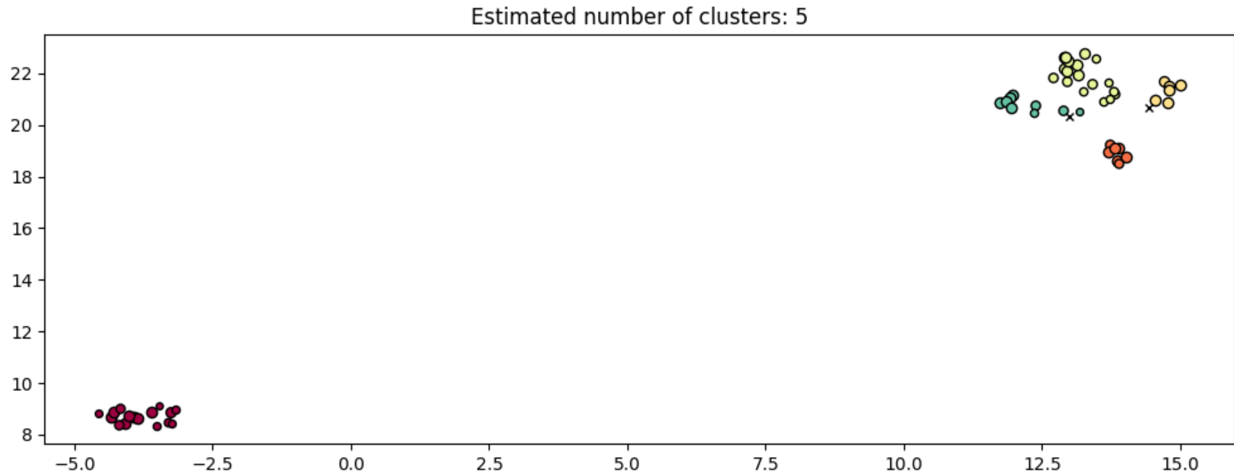

Figure 7: clusters determined by the HDBSCAN algorithm

To explore the relation between the outcome of the clustering algorithm and the input feature, we have developed a visualization function that colors the data points according to the value of a single feature, corresponding to a color scale. This heatmap gives us a better understanding of how clusters are formed, and which features are prominent (see *Figure 8*).

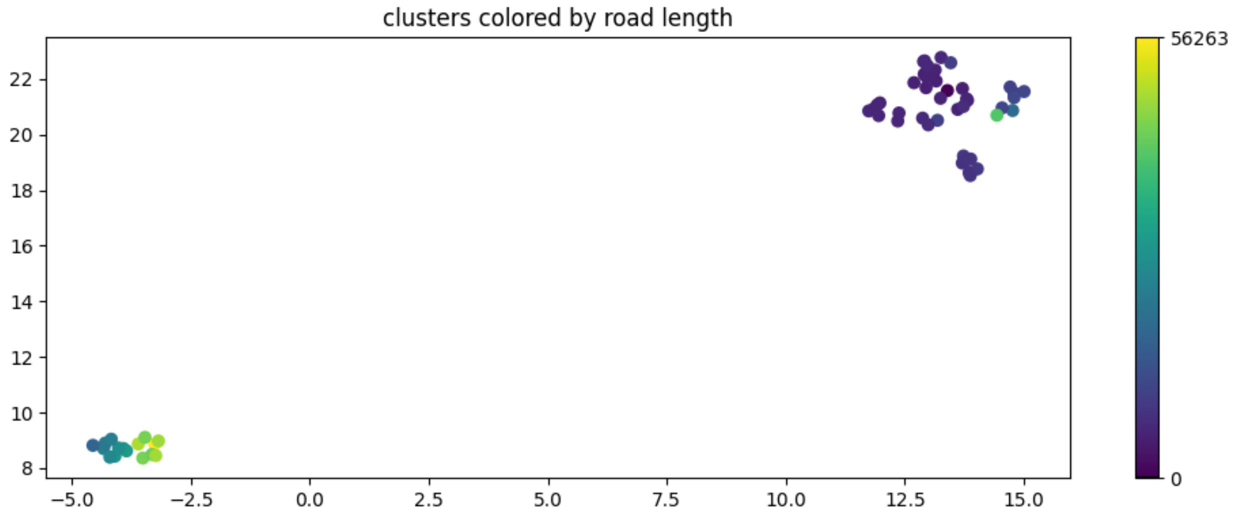

Figure 8: data points colored based on the value of the feature

Besides scatter plots, we have also constructed a method, based on SUMO visualization tools that creates an image of the traffic network with the roads colored based on the cluster they were assigned. This graph gives us a better understanding of the network by revealing possible connections between road segments (see *Figure 9*).

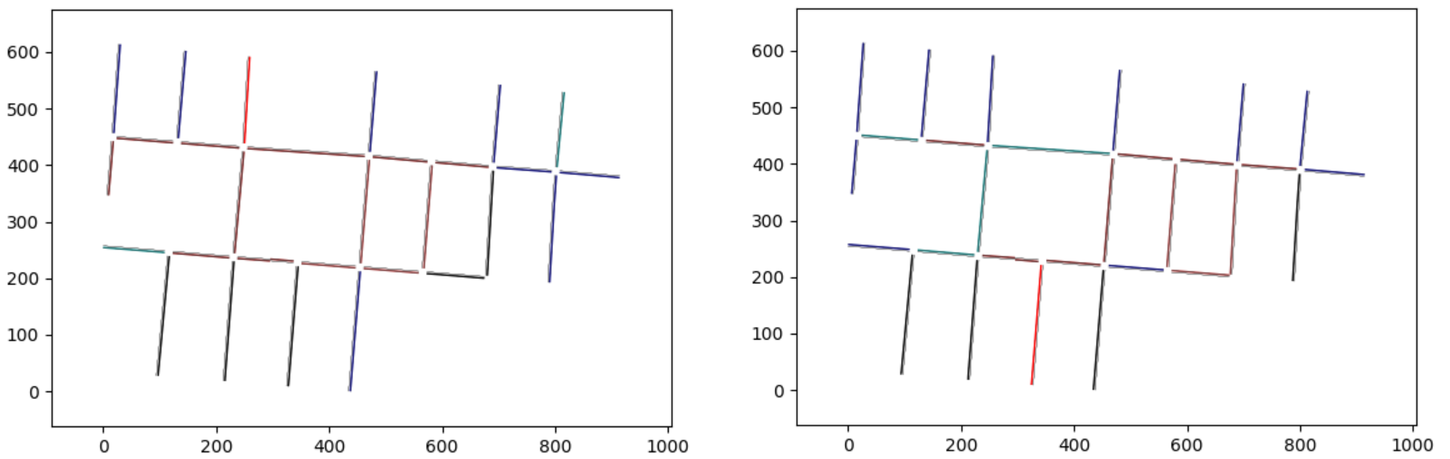

Figure 9: the traffic network with lanes colored based on their cluster (adjacent lanes colored separately)

6.2 Training prediction models

We used prediction models to forecast traffic flow parameters, such as mean travel time, flow, and a flag which shows that no vehicle passed the road. In both models we used hyperparameter tuning with scikit-learn's HalvingGridSearchCV [61]. There are three ways to train a model on our dataset. We can train a model on every road section, thus it can learn its structure, common underlying patterns, and expect the best performance with this method. However, it has high computational cost, since we need to train, store and evaluate a separate model for every road. This approach scales worse on larger networks, thereby it cannot be used on real urban environments without over-simplifying its road infrastructure. The opposite of this method is when only one model is trained on the whole network, so it learns the common traffic patterns, but it is not specified for one road, however the computation cost will be low. The third way, introduced by this paper is to train the model using the additional information coming from clustering. The idea behind clustering is that if we group roads with similar parameters their traffic patterns will probably be similar, thus the models can learn these specified patterns, while we need to train and use way less prediction models. In practise, all these methods can be converted into the clustering method. Separate roads means that every road section has its own cluster, while if we use one model for all roads means we have only one cluster, which contains every lane. Thus, we can use the same method, same code for all these methods as their pipeline are matching.

Both models predict traffic parameters 10 minutes in advance and use data from the previous 25 minutes. The pipeline has three main parts. First, we create the necessary input vectors for the model in each cluster from the aggregated data frame, split it into train and test sets, and rescale their values using scikit-learn's StandardScaler [62]. Normalization is necessary to eliminate the range differences between the features, which could affect the model negatively [63]. If a cluster contains multiple roads, then we mix their input vectors to use data from all roads in the cluster. In this case, we need to cut the mixed dataset to have the same length as one road's training dataset, so each method will use the same amount of data. Equal number of elements in the dataset provides fair comparison between the methods. Next, we train our model on the training dataset for every cluster and store the models. Finally, we evaluate the models on each road's test dataset with multiple metrics. For evaluate travel time and flow predictions we used MAE, MAPE, RMSE as these are regression metrics. The flag that shows whether a vehicle has passed the road is evaluated with binary classification metrics, such as accuracy, precision, recall. Since our models are built

for regression, they can't predict binary flags directly. The model predicts a continuous value, if it is below the threshold, it results false, else true. The mentioned threshold is optimized on the training dataset.

6.2.1 SVR

To implement Support Vector Regression as the prediction model, we used scikit-learn library [64]. For training SVR, we need to create its input vectors using sliding window technique, then train the model while tuning its hyperparameters.

To create the input vectors for SVR, we iterated through each cluster. If a cluster has multiple roads, we created the input vectors for each road, and mix them. Since SVR uses the given features for making prediction, we used sliding window technique to create these features. One window covers 25 minutes, which is five timeframes, since one timeframe is five minutes long. The output vector contains the ground truth predictions, which contains data from 10 minutes afterwards the current timeframe. This means that there are 15 features in our one-dimensional input vector, and three features in every output vector. In each cluster, we mix every road's input vectors, then split it to train and test datasets with scikit-learn's train_test_split function [65]. Every road has 1828 data vectors, so when we train every road segment separately, SVR has this amount of data to train. If a cluster contains n roads, then it has $n * 1828$ data vectors, which would give it advantage. To give equal chances for each method, we resample the training dataset to have a maximum of 1828 data vectors. After the resampling, StandardScaler is used to normalize the dataset. StandardScaler scales the dataset in a way that it has zero mean and unit variance. The transformation is shown in *Equation 6.1*.

$$z = \frac{x-\mu}{\sigma} \tag{6.1}$$

Next, we trained the SVR with the given training data. SVR has three main hyperparameters, which were mentioned in a previous section: C, epsilon, kernel function (*Chapter 3.3.1*). Since every cluster has different traffic patterns, their models probably need different hyperparameters. These parameters have high influence in the accuracy of the model, thus we need to tune them. For hyperparameter tuning, we used HalvingGridSearchCV from scikit-learn, which is much faster than normal GridSearchCV [66]. It starts evaluating all the candidates with small amount of data,

and in each iteration, it selects the best candidates, then increase the resources. To select the best candidates, we can use different metrics, such as negative mean absolute error, or negative mean squared error. Negative MSE is sensitive for outliers, since it squares the errors, and we find it less effective, outliers have too much influence on the selection. Negative MAE gave logical predictions with good results, so we decided to use that as the hyperparameter selection metric.

Finally, every cluster's model and scaler are stored with the cluster ID.

6.2.2 LSTM

One of the most common deep learning model for traffic prediction is Long Short-Term Memory, which is specialized for time series data. We used TensorFlow's Keras API [67] to build the deep learning model, and KerasRegressor [68] from SciKeras to make the regression model. To access GPU, we have run our code in Google Colab [69], which is a hosted Jupyter notebook service. Colab provides GPU free of charge, so it is well suited for machine learning, especially deep learning.

For LSTM, the process of creating the input vectors is similar as in SVR, which is described in the section above. We use the sliding window technique, split the data into train and test set, cut it if a cluster contains multiple roads, then scale it with StandardScaler. There is a big difference in the structure of the input vectors, because LSTM process the data sequentially. This means, that an input vector has two dimensions: timesteps, features. As we use data from the last five timeframes and one frame contains three features, the input vector has 5x3 shape. The output vectors contain the ground truth prediction values of all three features.

We tried multiple architecture around LSTM, and we concluded that one LSTM layer is enough, if we add more, it does not give any significant improvement in result, but increases the runtime. A dense layer with ReLU activation function before and after the LSTM layer gave better results,

and every layer contains 15 neurons. We visualized the final architecture with plot_model function from tf.keras.utils [70]. The architecture of the final LSTM network is shown on *Figure 10*.

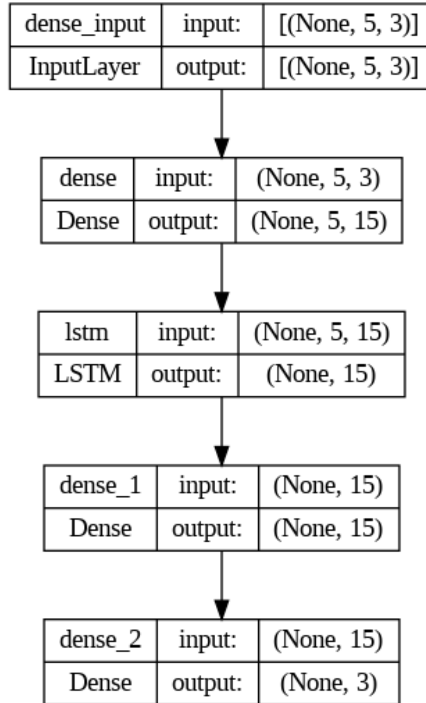

Figure 10: Visualize neural network architecture

We used hyperparameter tuning on the batch size and optimizer with HalvingGridSearchCV. The other parameters, such as loss, epochs, learning rate are not tuned, since it would cause long runtime, but we tried many variations, to get better hyperparameters. In SVR, we mentioned that the dataset can have outliers, and if we use negative MSE as the metric for hyperparameter tuning, it will have too much influence on the model's training (*Chapter 6.2.1*). This statement is especially important for LSTM, because if we chose mean squared error as the model's loss function, the output predictions are much worse, than with mean absolute error. Thus, we used MAE, which will be our main metric in the evaluation. *Table 1* shows the hyperparameters of the LSTM network.

Table 1: LSTM hyperparameters

Hyperparameter	Value
Batch size	32
Epochs	100
Loss	MAE
Optimizer	Adam
Learning rate	0.001

7 Results

To conclude the efficiency of the clustering, we need to evaluate the prediction models. We made the predictions for every road with their given test dataset and model, then evaluated them using different metrics. These metrics are MAE, RMSE, and MAPE for regression problems, while the binary flag is evaluated with accuracy, precision, recall, and weighted F1 score.

MAE averages the absolute differences between the real and the predicted value (*Equation 7.1*).

$$MAE = \frac{1}{n} * \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{7.1}$$

RMSE averages the squared errors, then takes its squared root (*Equation 7.2*). It is sensitive for outliers.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{n}} \tag{7.2}$$

MAPE averages the absolute percent differences between the real and the predicted value (*Equation 7.3*).

$$MAPE = \frac{1}{n} * \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{\hat{y}_i} \right| \tag{7.3}$$

Accuracy is the correct prediction divided by the number of all predictions (*Equation 7.4*).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.4}$$

Precision shows how good our predictions are on the positive predictions, its value is true positives divided by all the predicted positives (*Equation 7.5*).

$$Precision = \frac{TP}{TP + FP}$$

(7.5)

Recall shows how correctly we predict true positives. Its value is calculated by dividing the true positives with all the real positives (*Equation 7.6*).

$$Recall = \frac{TP}{TP + FN}$$

(7.6)

F1 score is the harmonic mean of precision and recall (*Equation 7.7*). Weighted F1 score is the weighted average of F1 scores of each class.

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

(7.7)

In this section, first, we evaluated the prediction results if every road has a separate forecasting model. The final error value for each metric is made by averaging the errors of every road. There is a road section, which produces the most of the final error values, because it is overloaded if traffic has higher density. To reduce the effect of outlier roads and have a real understanding of how the models perform on normal traffic, we also calculated the final error values by taking the median of the roads' prediction errors.

In the second part of this section, we evaluate the efficiency and runtime gain of our method, compared to other, simpler solutions. The comparison includes two methods beside ours, when there is a model for every road segment separately, and when there is one model for all roads. We show the benefits of clustering through the comparison, how it affects training time and prediction accuracy.

7.1 Evaluate without clustering

It is important to see how our models perform in general if there is a model for every road separately, to prove that the prediction models are working accurately. If roads have separate models, we expect the best performance, since it can learn the road section's specified traffic patterns. The final evaluation scores for each metrics can be calculated by average every road's

errors. With this approach the results are not seemed to be accurate, it produces higher errors than expected.

Table 2: Prediction errors with averaging

	SVR	LSTM
Travel time MAE	9.32	8.84
Travel time MAPE	27%	27%
Travel time RMSE	52.46	45.23
Flow MAE	32.73	32.16
Flow RMSE	45.74	44.91
Flag accuracy	95%	96%
Flag precision	93%	94%
Flag recall	95%	95%
Flag F1 score	94%	94%

As shown in Table 2, errors are higher than expected, this is particularly noticeable on the MAPE of the travel time, which is 27%. We did not calculate MAPE of the flow, because flows can be zero, which could sometime cause division by zero, which is undefined. However, flow has a mean of 132.88 secundum, thus its MAE and RMSE values are higher than expected.

We investigated the cause of the high errors and found that one specific road produces most of the errors, because it is always overloaded if traffic has higher density.

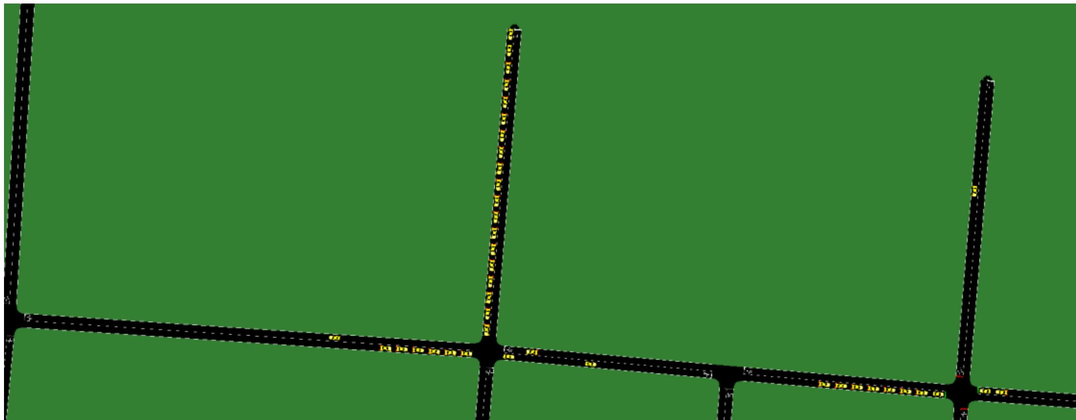

Figure 11: Overloaded road section

Figure 11 shows the overloaded road section, which happens because vehicles cannot enter the intersection, while too many vehicles enter the road. This road section has a mean travel time of 304.69 sec, while its MAE is 195.50 and its MAPE is 7.52. The reason behind these high errors is the randomness in the road's data, and the way we fill travel time if no vehicle passed the road. Vehicles in this road can hardly enter the intersection if traffic has high density, because other vehicles prevent them. Thus, it is difficult to predict how long does it take to go through the road, since it depends on the other roads' vehicles. The other problem, which causes the huge MAPE on this road, is the situation when no vehicle passes the road in the 5-minute frame, we calculate the travel time by the speed limit and road length. This approach is good in normal traffic, when there are no anomalies, because this usually happens when the road is empty, which means we can drive with the highest allowed speed, or in other word, with speed limit. The problem occurs when a road is overloaded, or an anomaly happens on it. Then it becomes possible that vehicles cannot enter the intersection for more than five minutes, so we detect no vehicles, while there are many vehicles on the road. When this happens, the road has sometimes huge travel time if at least one vehicle can enter the intersection, because it has waited in the congestion for a long time, and small travel time if no vehicle moves for 5 minutes, since we fill these data by the speed limit. In this outlier road section, this situation happens many times, which causes the bad error scores. *Figure 12* shows an example of how scattered travel time can be on this road.

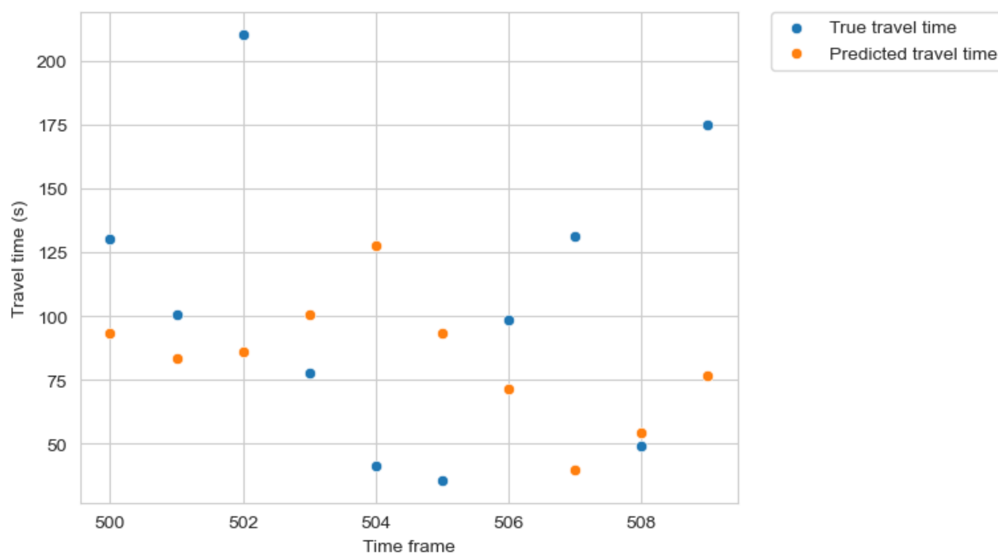

Figure 12: Overloaded road's true and predicted mean travel times

To give the outlier road less influence on the evaluation, we decided to get the median of the roads' errors. Median gives more realistic picture of how our models predict in normal traffic.

Table 3: Prediction errors with median

	SVR	LSTM
Travel time MAE	3.75	3.39
Travel time MAPE	8%	9%
Travel time RMSE	17.81	19.26
Flow MAE	25.94	25.81
Flow RMSE	34.74	34.48
Flag accuracy	99%	99%
Flag precision	98%	99%
Flag recall	99%	99%
Flag F1 score	98%	99%

Table 3 contains the results taking the median of every road's errors. The result shows that both our models are predicting accurately, with less than 10% MAPE on travel time, and MAE is low. RMSE is much higher than MAE, because our dataset contains scenarios with anomalies, which can produce outliers. Since RMSE is sensitive for outliers, because of squaring, it gives much higher value than MAE. On the classification metrics, models have 98% and 99% F1 score.

Overall, our prediction models are predicting accurately on normal traffic, but in cases, where no vehicle passed the road can create outliers, and scattered data if a road overloads. In this situation we cannot expect the model to predict correctly, as scattering is random, it depends on the vehicles of other roads. However, the problem can occur just in extreme traffic conditions, which is rare.

7.2 Evaluate clustering

As mentioned before (*Chapter 7*), there are three main methods for traffic forecasting, which we have compared. There can be a model for every road, or for each cluster, or just one prediction

model for the whole network. We expect the best performance, if we build a separate forecasting model for every road, so it can learn its specific traffic patterns. If there is just one prediction model for all roads, then it learns general traffic patterns, which we expect to be less accurate. Our method, which uses clustering can learn the roads' specific traffic patterns, since a cluster contains the roads with similar parameters, and it has much less runtime and computational cost, because we need to train less models.

Table 3: Differences between the methods; benchmark is when every road has a separate model

	With clustering SVR	One model for all SVR	With clustering LSTM	One model for all LSTM
Travel time MAE	+0.53%	+26.29%	+12.91%	+57%
Travel time MAPE	+14.90%	+109.43%	-4.65%	+144.59%
Travel time RMSE	+0.38%	+121.79%	+42.75%	+81.44%
Flow MAE	+3.45%	+5.55%	+1.6%	+4.76%
Flow RMSE	+0.12%	+1.69%	-0.6%	+5.21%
Flag accuracy	0%	0%	-0.33%	-0.11%
Flag precision	-0.08%	-0.77%	-0.44%	-1.32%
Flag recall	0%	0%	-0.22%	-0.22%
Flag F1 recall	0%	-0.17%	-0.44%	-0.66%

Table 3 shows the differences in percentage between the methods' errors, the benchmark is when every road has a separate prediction model, which we evaluated in the previous subsection. As we explained in the previous subsection, if we take the median of every road section's error, it gives more realistic picture of how our models predict in normal traffic, so we used median values for the comparison. The results confirm our prior expectations, as clustering does not produce much higher errors. In SVR, MAE and RMSE is higher by less than 1%, only travel time's MAPE is worse significantly than with using separate model for every road. With LSTM, differences are a bit higher, especially in the RMSE, however MAPE has reduced. Classification metrics show small decrease, but it is not significant. If we use one prediction model for all roads, our errors became much larger, thus it wouldn't be effective for forecasting.

We measured the runtime of the training with different methods on the free version of Google Colab (2023/10/23 release). If the prediction model is SVR, with using separate models, the runtime is 425 sec, with clustering it is 84 sec, which means that training is more than four times faster with clustering. If LSTM is the forecasting model, the difference is even larger, with separate models, the runtime is 2130 sec, while clustering reduces it to 271 sec, thus it is almost eight times less. Since clustering does not produce much higher errors, while runtime decreased by many times, we conclude that clustering was successful.

Table 5 shows that SVR and LSTM have similar performances if each cluster has a prediction model, LSTM gives slightly better predictions on the flow, and produces smaller MAE, but higher MAPE and RMSE then SVR in travel time. If we use clustering, the results are still similar, we can only observe significant difference in RMSE, where SVR gives smaller error. Since SVR had much smaller runtime, but similar performance, we draw the conclusion that machine learning models can be as efficient as deep learning models in some traffic forecasting problems.

Table 5: SVR and LSTM comparison

	SVR	LSTM
Travel time MAE	3.77	3.83
Travel time MAPE	10%	9%
Travel time RMSE	17.88	27.49
Flow MAE	26.84	26.23
Flow RMSE	34.78	34.38
Flag accuracy	99%	99%
Flag precision	98%	98%
Flag recall	99%	99%
Flag F1 score	98%	98%

8 Conclusion

In summary, our work concluded that grouping roads with similar parameters into clusters, and training separate prediction models for each cluster reduces computational cost, while remaining accurate. We used SUMO to create a traffic simulation with different scenarios to represent real life. We aggregated the raw dataset to get every road's parameters in 5-minute timeframes. Then we created the clusters with HDBSCAN, which groups roads by their parameters. For traffic prediction, we trained SVR and LSTM models on each cluster. The models were evaluated with and without clustering. The evaluation showed that our models are predicting accurately in general, however one road section produces huge errors, because it is overloaded, which makes the average errors of the roads high. With taking the median of the roads' prediction errors, the outlier has less influence, and it showed that both models are accurate in normal traffic. To evaluate the accuracy of the clustering, we compared our clustering method with two other forecasting solutions, as we can train a model for each road, or just one model for all roads. After comparing the three methods, we concluded that clustering does not produce much greater errors, while its runtime is decreased more than four times in SVR, and almost eight times in LSTM. Overall, these results prove the effectiveness of clustering. We also compared SVR and LSTM, they produced similar results, but SVR had a much smaller runtime.

In future works, out method can be improved. With generating more data from the simulation, models could learn more traffic patterns, making it more robust. It is possible that using more complex, hybrid models could give better prediction results. Using hyperparameter tuning on all parameters of the deep learning model could also make our model more accurate, since it helps to train better on the training dataset.

Acknowledgements

We would like to thank our supervisors Dr. Vilmos Simon and Norman Bereczki for providing immense help throughout this project. Their insights, advice and overall support made this work possible.

Bibliography

[1] N. Bereczki and V. Simon, “Machine Learning Use-Cases in C-ITS Applications,” *Infocommunications journal,* vol. 15, pp. 26-43, January 2023.

[2] M. Shaygan, C. Meese, W. Li, X. Zhao and M. Nejad, “Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities,” *Transportation Research Part C: Emerging Technologies,* vol. 145, p. 103921, 2022.

[3] D. Schrank, B. Eisele, T. Lomax and others, “Urban mobility report 2019,” 2019.

[4] S. R. Samal, M. Mohanty and S. M. Santhakumar, “Adverse Effect of Congestion on Economy, Health and Environment Under Mixed Traffic Scenario,” *Transportation in Developing Economies,* vol. 7, p. 15, 2021.

[5] H. Boogaard, A. P. Patton, R. W. Atkinson, J. R. Brook, H. H. Chang, D. L. Crouse, J. C. Fussell, G. Hoek, B. Hoffmann, R. Kappeler, M. K. Joss, M. Ondras, S. K. Sagiv, E. Samoli, R. Shaikh, A. Smargiassi, A. A. Szpiro, E. D. S. V. Vliet, D. Vienneau, J. Weuve, F. W. Lurmann and F. Forastiere, “Long-term exposure to traffic-related air pollution and selected health outcomes: A systematic review and meta-analysis,” *Environment International,* vol. 164, p. 107262, 2022.

[6] F. An, J. Liu, W. Lu and D. Jareemit, “A review of the effect of traffic-related air pollution around schools on student health and its mitigation,” *Journal of Transport & Health,* vol. 23, p. 101249, 2021.

[7] “U.S. Greenhouse Gas Emissions and Sinks 1990–2021,” Agency, U.S. Environmental Protection, 2023.

[8] S. Bharadwaj, S. Ballare, Rohit and M. K. Chandel, “Impact of congestion on greenhouse gas emissions for road transport in Mumbai metropolitan region,” *Transportation Research Procedia,* vol. 25, pp. 3538-3551, 2017.

[9] S. Neelakandan, M. A. Berlin, S. Tripathi, V. B. Devi, I. Bhardwaj and N. Arulkumar, “IoT-based traffic prediction and traffic signal control system for smart city,” *Soft Computing,* vol. 25, p. 12241–12248, 2021.

[10] M. Akhtar and S. Moridpour, “A review of traffic congestion prediction using Artificial Intelligence,” *Journal of Advanced Transportation,* Vols. 1-18, Jan. 2021.

[11] B. Priambodo, A. Ahmad and R. A. Kadir, “redicting traffic flow propagation based on congestion at neighbouring roads using hidden Markov model,,” *IEEE Access,* vol. 9, p. 85933–85946, Jan. 2021.

[12] S. V. Bino, A. Unnikrishnan and B. Kannan, “Gray Level Co-Occurrence Matrices: Generalisation and some new features,” *arXiv,* May 2012.

[13] A. Ng, M. Jordan and Y. Weiss, “On Spectral Clustering: Analysis and an algorithm,” 2001.

[14] M. A. Mondal and Z. Rehena, “Identifying Traffic Congestion Pattern using K-means Clustering Technique,” Apr. 2019.

[15] M. Ahmed, R. Seraj and S. M. S. Islam, “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation,” *Electronics,* vol. 9, p. 1295, Aug. 2020.

[16] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” *Project Euclid,* Jan. 1967.

[17] H. Shenghua, N. Zhihua and H. Jiaxin, “Road traffic congestion prediction based on random Forest and DBSCAN combined model,” *2020 5th International Conference on Smart Grid and Electrical Automation (ICSGEA),* Jun. 2020.

[18] B. Medina-Salgado, E. Sanchez-DelaCruz, P. Pozos-Parra and J. E. Sierra, “Urban traffic flow prediction techniques: A review,” *Sustainable Computing: Informatics and Systems,* vol. 35, p. 100739, 2022.

[19] G. Lin, A. Lin and D. Gu, “Using support vector regression and K-nearest neighbors for short-term traffic flow prediction based on maximal information coefficient,” *Information Sciences,* vol. 608, p. 517–531, 2022.

[20] Z. E. Shen, “Long-time traffic speed prediction model based on XGBoost,” 2023.

[21] M. A. Mondal and Z. Rehena, “Stacked LSTM for Short-Term Traffic Flow Prediction using Multivariate Time Series Dataset,” *Arabian Journal for Science and Engineering,* vol. 47, p. 10515–10529, 2022.

[22] S. Wang, C. Shao, J. Zhang, Y. Zheng and M. Meng, “Traffic flow prediction using bi-directional gated recurrent unit method,” *Urban informatics,* vol. 1, p. 16, 2022.

[23] A. Almeida, S. Brás, I. Oliveira and S. Sargento, “Vehicular traffic flow prediction using deployed traffic counters in a city,” *Future Generation Computer Systems,* vol. 128, p. 429–442, 2022.

[24] V. Rajalakshmi and S. Ganesh Vaidyanathan, “Hybrid CNN-LSTM for traffic flow forecasting,” in *Proceedings of 2nd International Conference on Artificial Intelligence: Advances and Applications: ICAIAA 2021*, 2022.

[25] Z. Ling-yun, H. Ying, Z. Kai, L. Hai-peng and D. Yu-jie, “Short-term Traffic Flow Prediction Model Based on Deep Learning,” *Computer and Modernization,* p. 54, 2022.

[26] S. Reza, M. C. Ferreira, J. J. M. Machado and J. M. R. S. Tavares, “A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks,” *Expert Systems with Applications,* vol. 202, p. 117275, 2022.

[27] S. P. Kailasam, K. Aruna, M. M. J. I. Sathik and others, “Traffic flow Prediction with Big Data Using SAES Algorithm,” *JCSMC,* vol. 5, p. 186–193, 2016.

[28] Y. Djenouri, A. Belhadi, G. Srivastava and J. C.-W. Lin, “Hybrid graph convolution neural network and branch-and-bound optimization for traffic flow forecasting,” *Future Generation Computer Systems,* vol. 139, p. 100–108, 2023.

[29] B. Lindemann, T. Müller, H. Vietz, N. Jazdi and M. Weyrich, “A survey on long short-term memory networks for time series prediction,” *Procedia CIRP,* vol. 99, p. 650–655, 2021.

[30] H. B. Barlow, “Unsupervised learning,” Neural Computation, vol. 1, Sep. 1989, p. 295–311.

[31] K. J. Cios, R. W. Swiniarski, W. Pedrycz and L. A. Kurgan, “Unsupervised Learning: association rules”, 2007, p. 289–306.

[32] L. van der Maaten, E. O. Postma, H. J. van den Herik and others, Dimensionality reduction: A comparative review. Journal of Machine Learning Research, 2009, p. 13.

[33] T. S. Madhulatha, An overview on clustering methods, May 2012.

[34] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” p. 226–231, 1999.

[35] J. Tian, Z. Yu, L. Liu, W. Wu, H. Zhu and X. Liu, “An abnormal traffic detection method in smart substations based on coupling field extraction and DBSCAN,” *E3S Web of Conferences,* p. 260:02005, 2021.

[36] R. J. G. B. Campello, D. Moulavi and J. Sander, “Density-Based clustering based on hierarchical density estimates,” *Lecture Notes in Computer Science,* p. 160–172, 2013.

[37] B. Costa, J. Freire, H. Cavalcante, M. Homci, A. Castro, R. Viégas Jr, B. Meiguins and J. Morais, “Fault classification on transmission lines using KNN-DTW,” *Lecture Notes in Computer Science,* pp. 174-187, 2017.

[38] M. Ghanbari and M. Goldani, “Support vector regression parameters optimization using Golden Sine algorithm and its application in stock market,” *arXiv preprint arXiv:2103.11459,* 2021.

[39] T. Kleynhans, M. Montanaro, A. Gerace and C. Kanan, “Predicting Top-of-Atmosphere Thermal Radiance Using MERRA-2 Atmospheric Data with Deep Learning,” *Remote Sensing,* vol. 9, p. 1133, November 2017.

[40] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena,* vol. 404, p. 132306, 2020.

[41] "Dive Into Deep Learning," [Online]. Available: https://d2l.ai/chapter_recurrent-modern/lstm.html.

[42] G. Leduc, "Road traffic data: Collection methods and applications," *Working Papers on Energy, Transport and Climate Change,* pp. 1-55, 2008.

[43] M. Behrisch, L. Bieker-Walz, J. Erdmann and D. Krajzewicz, "SUMO – Simulation of Urban MObility: An Overview," *ResearchGate,* Oct, 2011.

[44] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner and E. Wiessner, "Microscopic Traffic Simulation using SUMO," 2018.

[45] "SUMO documentation," 2023. [Online]. Available: https://sumo.dlr.de/docs/index.html.

[46] "Artery documentation," 2023. [Online]. Available: http://artery.v2x-research.eu/#license.

[47] "OSMWebWizard documentation," 2023. [Online]. Available: https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html.

[48] "Project Jupyter Documentation," [Online]. Available: https://docs.jupyter.org/en/latest/.

[49] "pandas documentation," 20 09 2023. [Online]. Available: https://pandas.pydata.org/docs/index.html.

[50] "NumPy documentation," [Online]. Available: https://numpy.org/doc/stable/index.html.

[51] M. Bauder, A. Festag, T. Kubjatko and H.-G. Schweiger, "Data Accuracy in Vehicle-to-X Cooperative Awareness Messages: An Experimental Study for the First Commercial Deployment of C-Its in Europe," *Available at SSRN 4442746.*

[52] "Valhalla Docs," [Online]. Available: https://valhalla.github.io/valhalla/meili/overview/.

[53] N. Doulos, "Nick Doulos," 05 04 2023. [Online]. Available: https://nickdoulos.com/posts/map-matching/.

[54] E. N. ETSI ETSI, "302 637-2 V1. 3.1 Intelligent Transport Systems (ITS)," *Vehicular Communications,* p. 1–44.

[55] "geojson.io," [Online]. Available: https://geojson.io/.

[56] "The Shapely User Manual," 12 10 2023. [Online]. Available: https://shapely.readthedocs.io/en/stable/manual.html.

[57] "Welcome to GeoPy's documentation!," [Online]. Available: https://geopy.readthedocs.io/en/stable/.

[58] S. Aghabozorgi, A. S. Shirkhorshidi and T. Y. Wah, "Time-series clustering – A decade review," *Information Systems,* vol. 53, pp. 16-38, Oct. 2015.

[59] "UMAP documentation," 2023. [Online]. Available: https://umap-learn.readthedocs.io/en/latest/.

[60] "HDBSCAN documentation - parameter selection," 2023. [Online]. Available: https://hdbscan.readthedocs.io/en/latest/parameter_selection.html.

[61] "sklearn.model_selection.HalvingGridSearchCV," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingGridSearchCV.html.

[62] "sklearn.preprocessing.StandardScaler," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[63] X. Wan, "Influence of feature scaling on convergence of gradient iterative algorithm," in *Journal of physics: Conference series*, 2019.

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

[65] "sklearn.model_selection.train_test_split," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[66] "Tuning the hyper-parameters of an estimator," [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html#successive-halving-user-guide.

[67] "Keras: The high-level API for TensorFlow," 8 6 2023. [Online]. Available: https://www.tensorflow.org/guide/keras.

[68] "scikeras.wrappers.KerasRegressor," [Online]. Available: https://adriangb.com/scikeras/stable/generated/scikeras.wrappers.KerasRegressor.html.

[69] "Colab Enterprise documentation," [Online]. Available: https://cloud.google.com/colab/docs.

[70] "tf.keras.utils.plot_model," 04 10 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model.

[71] "DTAIDistance documentation," [Online]. Available: https://dtaidistance.readthedocs.io/en/latest/index.html. [Accessed 2023].