POLITECNICO DI TORINO

Master degree course in Ingegneria Informatica
(Computer Engineering)

Master Degree Thesis

Traffic Load Estimation from Structural Health Monitoring Sensors using Supervised Learning

Supervisors
Doc. Daniele Jahier Pagliari
Doc. Alessio Burrello

Candidate
Giovanni ZARA
matricola: 252735

ACADEMIC YEAR 2020-2021

Summary

In the last decades there has been an evolution in the transport system, both for carriage of passengers and goods. In fact, the use of vehicles has become essential in the life of most of the population.
This evolution has led to an increasing number of vehicles in circulation with the consequent increase in traffic congestion in travel times, road accidents and environmental problems related to pollution, especially near the large cities where urban traffic is often related to goods transport.
Various Traffic Load Estimation (TLE) approaches have been implemented, which are essential for analyzing and managing vehicular traffic in the critical nodes of the road infrastructure.
The standard approaches of TLE are based on the installation of dedicated sensors such as cameras or infrared sensors or by examining data from sensors present in smartphones.
These types of approaches have various limitations, such as the large costs to install and maintain dedicated sensors or the need for user collaboration for smartphone-based approaches.
Approaches based on the analysis of data from sensors, such as accelerometers, already present in the critical points of the road infrastructure for Structural Health Monitoring (SHM) have recently been studied.
In this category, the previous solutions identify and count vehicles by detecting anomalies without using labeled datasets.
In this thesis, a TLE system is proposed using a supervised learning approach based on the data collected by the sensors of a SHM installation present in a bridge in Italy.
The datasets used in this study are accelerations grouped into time windows of various sizes. For each time window there are two labels corresponding to the count of light vehicles (such as cars) and the count of heavy vehicles (such as trucks) that have crossed the section of the bridge examined.
These datasets are used to train Machine Learning (ML) models trained as

regressors whose task is to estimate the count of light vehicles and heavy vehicles corresponding to each input time window.
In order to carry out this task, various models of both classic ML and Deep Learning were tested and compared.
From the experiments, the model that obtained the highest accuracy in the prediction of heavy vehicles was the Support Vector Regressor (SVR), which obtained a Mean Absolut Percentage Error (MAE%) of 7.6% and an R2 score of 0.97 in predicting heavy vehicles. On the other hand, the highest accuracy in the prediction of light vehicles was obtained by the K-Nearest Neighbors (KNN), which obtained a MAE% of 9% and an R2 score of 0.92.

Acknowledgements

While aware that this is not an end point, but a starting point, today marks the closing of an important chapter in my life. Studying Engineering is a winding and often uphill road, but I have been fortunate to face it surrounded by people who have continually motivated and supported me.
First of all I would like to thank my family, my father Roberto, my mother Francesca and my sisters Silvia and Sofia, for having always believed in me, supporting me even in the most difficult moments.
I thank my aunts and grandparents, for supporting me, allowing me to face every step of the way with the utmost serenity.
Finally, a big thank you to my supervisors, Daniele Jahier Pagliari and Alessio Burrello, for helping me with passion and great competence to solve every problem I faced in my thesis work. It was really a pleasure and a luck to work under your supervision.

Table of Contents

List of Tables

List of Figures

Acronyms

ADAM adaptive moment estimation.

AI Artificial Intelligence.

ANN Artificial Neural Network.

BPTT backpropagation through time.

CNN Convolutional Neural Network.

DL Deep Learning.

DNN Deep Neural Network.

DT Decision Tree.

FFNN Feed-Forward Neural Networks.

FPS frames per second.

GPU Graphic Processing Unit.

GRU Gated Recurrent Unit.

IoT Internet of Things.

KNN K-Nearest Neighbors.

LR Linear Regresso.

LSTM Long-Short Term Memory.

MAD Median Absolute Deviation.

MAE Mean Absolute Error.

ML Machine Learning.

MLP MultiLayer Perceptron.

MSE Mean Squared Error.

NAG Nesterov accelerated gradient.

NN Neural Network.

RBF Gaussian radial basis function.

ReLU Rectified Linear Unit.

RMS Root Mean Square.

RMSprop Root Mean Square Propogation.

RNN Recurrent Neural Network.

SGD Stochastic Gradient Descent.

SHM Structural Health Monitoring.

SimpleRNN Standard Recurrent Neural Network.

SVM Support Vector Machine.

SVR Support Vector Regressor.

tanh Hyperbolic Tangent Function.

TCN Temporal Convolutional Network.

TLE Traffic Load Estimation.

Chapter 1

Introduction

In the last century, the growth of the vehicle fleet has been exponential, with a consequent increase in traffic and interest in its monitoring. Monitoring the traffic load has several objectives. For instance, the collected data can support actions that increase the safety of drivers (tackle road traffic offences, driving assistance, information to drivers) or to support the institutions that deal with road management, which, through these systems, are facilitated in the management of traffic and in the planning of maintenance interventions. In addition, in recent years, traffic monitoring and management have seen a growing interest in relation to air pollution. Noteworthy, vehicular traffic is among the main causes of air pollution in urban areas.
There are many proven techniques and technologies offered by the industry that can be used for building a Traffic Load estimation (TLE) system. These techniques can be distinguished in manual TLE and automatic TLE.
The manual methods of traffic data acquisition are based on the detection performed by a human operator [1]. This approach has the advantage of having excellent accuracy. These methods have obvious disadvantages both in terms of data recording speed, of costs and of human resources.
The need to continuously and accurately collect traffic data, over long periods and at all hours, both day and night, has led the technological research of the sector to produce tools and increasingly refine automatic TLE systems. As with many other tasks, the Internet of Things paradigm (IoT), which involves the use of communicating sensors, is the basis of the growing diffusion of automatic TLE systems. In fact, automatic TLE systems, based on the IoT paradigm, use various types of sensors to collect information on the passage of vehicles. The data collected by the sensors is encoded,

transmitted and finally processed through various algorithmic techniques.
There are various approaches to implementing TLE systems. These approaches differ in the types of sensors used and consequently in the algorithmic techniques adopted.
Analyzing the state-of-the-art TLE systems (section 3.2), we can see that most of the cases can be divided in two families of approaches.
The first family of solutions, widely used, consists in the installation of dedicated sensors in the monitored road infrastructure.
These sensors, such as cameras [2], magnetodynamic sensors [3], infrared sensors [4] etc., achieve high accuracy in the counting and classification of vehicles. On the other hand, these types of approaches involve high costs of installing and maintaining the sensor nodes.
The second family of solutions, widely used, is based on the analysis of information collected by sensors already available in the drivers' smartphones [5] [6].
This solution reduces the cost of the hardware to zero. On the other hand, its efficiency is closely related to the collaboration of users in sharing their data. This type of approach achieves good performance only in monitoring areas crossed by the same group of drivers, who have agreed to share their data.
Recently a third family of approaches has achieved a growing interest. These approaches consist in implementing TLE using sensor networks, such as accelerometers [7], already present in the road infrastructure for Structural Health Monitoring (SHM) purposes.
The SHM consists in the automatic monitoring of large infrastructures such as bridges or buildings. The goal of SHM is to identify anomalous behaviors that could be signs of wear, and allows prompt and targeted maintenance interventions.
Typically when installed on bridges, SHM systems are composed of networks of accelerometers that measure the bridge's vibrations. The data collected by these accelerometer networks can be used to implement a TLE system by examining the vibrations of the bridge due to the vehicles crossing it.
This thesis proposes a TLE system based on the data collected by a SHM system present in Italy, described in section 3.1.
The purpose of this system is to calculate the number and category of vehicles crossing the viaduct in various time windows. This task is performed by the pipeline described in section 4.
Firstly, this pipeline starting from the labeled dataset described in section

4.1.1 apply a preprocessing phase (section 4.1). In this step, the acceleration of the starting dataset are grouped into various time windows. For each windows the labels corresponding to the number of heavy vehicles and of light ones crossing the bridge are calculated. After the windowing step several statistical features are extracted from the new dataset (section 4.1.3). The features extraction step produces a new dataset, where for each time window are present the extracted features and the labels.
Finally the new labelled datasets are used to train and test both supervised classic Machine Learning (ML) models and Deep Learning models to perform the regression, (section 4.2).
To the best of our knowledge, this work is the first that frame the problem of TLE based on accelerometers for SHM in a supervised way.
The results of each ML regressor tested, are reported and compared in section 5. From the results, it is evident that all models perform better in predicting the heavy vehicle count than the light vehicle count. The results also show that the performance of the models improve with increasing the time window under consideration. In the time windows of size of 60 seconds all the classic ML regressors achieve a $MAE\% \leq 10\%$ and a $R^2 \geq 0.95$. Considering all the window sizes we can see that the SVR is the model that overall performs best in the prediction of heavy vehicles. In the prediction of the light vehicle count, KNN was the model that obtained the best performance, obtaining a MAE% of 9% and an R^2 score of 0.92 for the windows with a size of 60 seconds.

.

Chapter 2

Background

2.1 Machine Learning

The scientific field of **Machine Learning (ML)** is a branch of **Artificial Intelligence (AI)**, as defined by Computer Scientist and machine learning pioneer [8] Tom M. Mitchell:

"Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience" [9].

Tom Mitchell provides a more detailed definition:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Machine Learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks [10].
In contrast to the traditional programming paradigm, where a programmer develops a set of rules or program, feeds it to the computer, and observes the output it produces, Machine Learning is about letting the computer figuring out such complex input-output relationships building a model that can make predictions or decisions [11].
Machine learning approaches are traditionally divided into three categories,

depending on the type of data available to the learning system:

- **Supervised Learning**: The goal of the supervised learning is to predict a target value (label) given input observation (features).
- **Unsupervised Learning**: No labels are given to the learning algorithm, leaving it on its own to find correlation, discovering patterns, in the provided input data.
- **Reinforcement learning**: is the training of machine learning models to make a sequence of decisions. A computer program interacts with a dynamic environment in which it must achieve a goal. As it navigates its problem space, the program is provided feedback that gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

In this thesis, various Supervised Learning algorithms have been used to perform the regression analysis of the number of vehicles that cross the checked bridge section in each time window. Therefore in this chapter the main characteristics of supervised learning and the models used will be illustrated.

2.2 Supervised Learning

Supervised learning is the most common subcategory of machine learning. Its scope is the prediction of a target variable, given a series of input observations. In machine learning, the model inputs are called "features" while the target values that supervised models are trained to predict are called labels. Supervised Learning uses a training set to teach models to predict the desired output. This training dataset includes inputs features (x) and correct output labels (y), which allow the model to learn over time the mapping function from the input to the output.

$$y = f(x) \tag{2.1}$$

In the training phase, the internal parameters of the model are adjusted to approximate the function $f()$. The algorithm measures its accuracy through

the loss function.
In this way, when you have new unseen input data x, you can predict its associated label with high accuracy. The main supervised learning problems are “regression” and “classification”.

2.2.1 Classification

A classification problem occurs when an object needs to be assigned into a predefined group or class based on a number of observed attributes and features related to that object. It can be a binary classification problem (two classes) or a multi-class problem (more than two classes). This problem could be solved by classification algorithms that take an input value and assign it a class, or category, that it fits into based on the learned mapping function. A common example of classification is determining if an email is spam or not. The algorithm will be given training data with emails that are both spam and not spam. The model will find the features within the data that correlate to either class and create the mapping function (equation 2.1) mentioned earlier. Then, when provided with an unseen email, the model will use this function to determine whether or not the email is spam [12].

2.2.2 Regression

Regression analysis is a sub field of supervised machine learning. It is a predictive statistical process where the model tries to find the relationship between a certain number of input values (features) and a continuous target variable (label). A regression problems occurs when we try to predict a continuous number, like predicting the prices of a house or a test score. For instance if we want to determine a student’s test grade based on how many hours they studied the week of the test, we can see in Figure 2.1 that there is a clear positive correlation between hours studied (feature) and the student’s final test score (label).
The goal of the regression model is to find the function that best fit the correlation between the hours studied and the test score in order to use this equation to predict the test score given a new input.
This is an example of simple linear regression. In fact there is only one feature in the data.
There are many types of more advanced regression algorithms. Each one tries to find the input/output correlation in a different way, using different

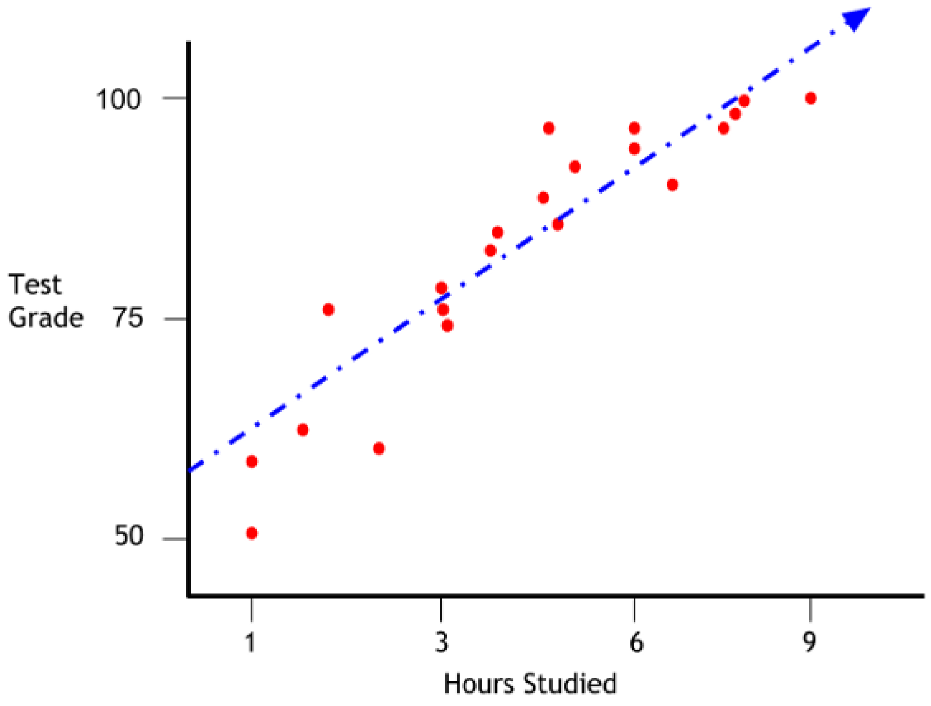


Figure 2.1: Correlation hours studied / test score, Image taken from [12]

type of mathematical and statistical function.

2.3 Deep Learning

Deep Learning (DL), is a sub-field of **Machine Learning**, which is based on more complicated (deep) models. Deep learning models, such as deep artificial neural networks, use multiple processing layers to discover patterns corresponding to different levels of abstraction. Each level learns a concept from the data on which subsequent level are based. The higher the level, the more abstract the concepts it learns.
In contrast with traditional ML methods, Deep Learning methods do not demand an advanced data pre-processing or feature extraction phase, generalizing their learning to all the fields with a reduced necessity of domain knowledge. In fact in traditional ML methods, performance is strongly

correlated to the feature engineering process, a complex process in which features are extracted. This process is time consuming and it requires domain knowledge. This problem is solved in DL through the so-called representation learning where different layers are used to learn the representation of the features starting from raw inputs. The main reason why Deep Learning techniques have received great attention only in recent decades, it is certainly due to progress in hardware, with the availability of new units of processing, such as graphics processing units (GPUs).
The basic functional unit of neural networks, is the artificial neuron. Artificial neurons are inspired by biological neurons roughly trying to model their structure and simulate their basic functions.
The formula describing the most basic type of artificial neurons is:

$$y = h(\sum_{i=1}^{n} w_i x_i + b) \tag{2.2}$$

Where x_i are the inputs of the neuron, multiplied to the respective weights w_i, b is a bias term and $h()$ denotes a non-linear function, usually called *activation function*, which is applied to the weighted sum.

2.4 Popular DNNs Architectures

Based on the types of layers and their connections, different families of Neural Networks can be identified, each of which has been developed to best perform a set of tasks.
Feed-Forward Neural Networks (FFNN) are networks made up of groups of artificial neurons called layers.
In these networks there is an input layer, one or more hiden layers and an output layer, the main feature of these networks, which differentiates them from the RNN, is that the flow of information travels in a single direction, starting from the input to get to the output layer without forming cycles.
Feed-forward networks do not have input memory occurred at previous times, so the output is determined only by the current input. This feature makes them unsuitable for processing sequential information of data.

2.4.1 Recurrent Neural Network (RNN)

Recurrent Neural Networks are a class of Artificial Neural Networks that achieve excellent performance in the analysis of sequential data. For this reason RNNs are commonly used in solving problems such as natural language processing, speech recognition, sentiment analysis, signal analysis, etc.
Like the architectures previously described, also the RNNs are made up of layers of artificial neurons, and are trained in the training phase to learn the correlation between inputs and outputs, but they differ from feedforward architectures in various aspects.

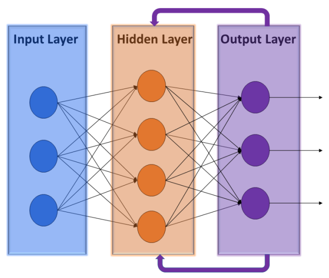


Figure 2.2: Recurrent Neural Network architecture.

In Figure 2.2 we can see a key feature of RNNs, from which the term recurrent comes, is that the information does not flows only forward, as happens in feedforward, but also backwards towards lower level layers.
This interconnection between layers allows the use of some of the layers as state memory, so unlike feedforward networks where the output depends only on the current input, in the RNN the same input can generate different outputs based on the inputs previously received [13].
The following equations define how a vanilla RNN evolves over time:

$$\boldsymbol{y}_t = \boldsymbol{f}(\boldsymbol{h_t}; \boldsymbol{\theta}) \tag{2.3}$$

$$\boldsymbol{h}_t = \boldsymbol{g}(\boldsymbol{h_{t-1}}; \boldsymbol{x_t}; \boldsymbol{\theta}) \tag{2.4}$$

where y_t is the output of the RNN at time t, x_t is the input to the RNN at time t, and h_t is the state of the hidden layer(s) at time t.
The equation 2.3 shows that, given the θ parameters (the weights and biases for the network), the output at time t depends only on the state of the

hidden layer at time t, just like a feedforward neural network. On the other hand, the hidden layer at time t depends both on the hidden layer at time $t-1$ and on the input on time t. This second equation demonstrates that the RNN output at time t is affected by the calculations made at time $t-1$ [14].
In the equations 2.3 and 2.4 we can see another distinctive feature of the RNN, that is the sharing of parameters on each level of the network. In feedforward networks the weights are different on each node while the recurrent neural networks share the same weight parameter within each layer of the network thus allowing the network to examine input vectors with variable length, not known a priori, such as the time series in which the lengths differ and are not always known.

Chapter 3

Related Works

The purpose of this thesis is to test the performance of various Machine Learning and Deep Learning algorithms in traffic load estimation (TLE) with various time windows. The studied networks were trained using supervised learning algorithms, using a labeled dataset. This dataset is made up of accelerations coming from a real Structural Health Monitoring (SHM) system, described in section 3.1 and of labels coming from a TLE system [15] [16] briefly described in section 3.2.4. In this section, in addition to describe the two aforementioned systems of interest for this thesis, various TLE systems will be briefly described. We will see that these systems collect data using various types of hardware (sensors, transmitters and receivers) and analyze them using different types of algorithms.

3.1 Structural Health Monitoring(SHM)

The Structural Health Monitoring is a branch of the Internet of Things (IoT) that have the objective to collect and process continuous information about the "Health" of structures such as bridges and buildings. SHM processes involve data acquisition about the structure's condition, through a specific set of sensor, storage and transmission systems and specific algorithms of Data interpretation and analysis. This Thesis is based on data provided by an SHM system installed in a highway bridge located in Italy. An high-level overview of the bridge is given in Figure 3.1.

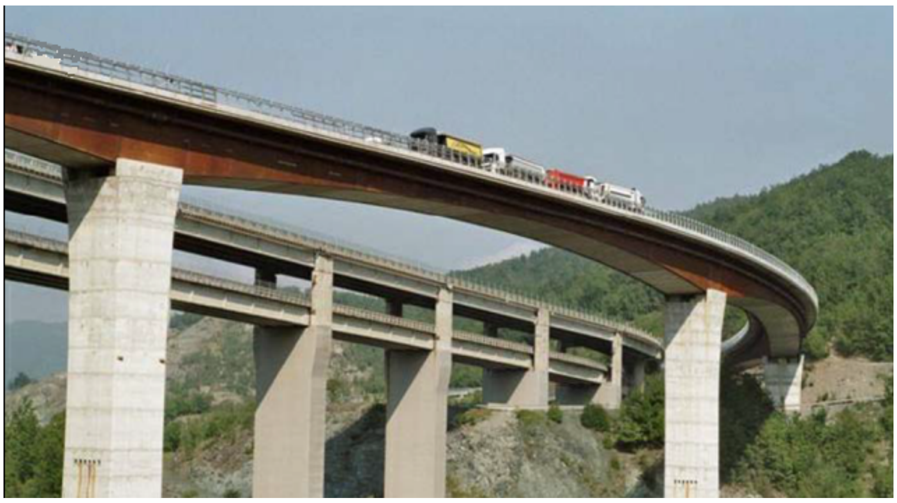


Figure 3.1: Overview of the bridge

3.1.1 SHM Installation

The viaduct is a composite box girder in which externally prestressed tendons were used to strengthen the structure. The total length of the viaduct is 580m and the main girder cross section height varies from 6.0m (at the bearings) to 3.0m (on the center-line of each span). The external tendons were equipped with 90 MEMS triaxial accelerometers, which measure the acceleration in three orthogonal directions (x; y; z), in particular 2 for each monitored element. Sensors were placed at the top of the prestressing cables in 10 different cross sections. Note that this kind of installation is not optimal for the monitoring of the vehicular traffic, which would benefit from sensors installed directly in the road infrastructure. The number of sensors for each equipped section is not constant along the viaduct. Each section can have from six to twelve sensors that register accelerations under traffic action. The data flow provided by sensors is sampled with a rate of 100 Hz in order to make the data streaming manageable by the network. In fact once the data is filtered, the sensor node transmits it to a local gateway through CAN-bus which support a maximum bit rate of 250 Kbps. Considering the bit rate of each node:

$$Rb = Nax * fs * Ls$$

where $Nax = 3$ is the number of axes of each sensor, $fs = 100Hz$ the sampling frequency and $Ls = 16$ is the number of bit for each sample. Each gateway can manage the data flow of maximum **50 sensors**. The system comprises two gateways, each one connected to 45 sensors. The data collected by the gateways are sent via Ethernet to the *"Ubiquity Nano M5"* station located halfway between the viaduct ends. M5 station is also connected via 5 GHz point-to-point Wi-Fi to the access point, which transfers the whole data to the cloud. The acceleration data, is stored in a cloud monitoring infrastructure, which allows real-time access and analysis [15][17].

3.2 Automatic Traffic Load Estimation (TLE)

Automatic traffic load estimation (TLE) is about detecting, counting and (possibly) classifying vehicles in transit on public road infrastructures, such as freeways, bridges or city streets.
The information extracted from TLE systems can improve several aspects related to sustainability in urban and suburban road infrastructures.
Use cases reported in recent work include scheduling maintenance interventions [7], monitoring and reducing air pollution [18], and forecasting the economic impact of travel delays due to traffic congestion [19].
As mentioned in Chapter 1, in recent works various TLE approaches have been proposed that differ mainly in the types of sensors used to collect the data and the types of algorithms used to analyze them. This is due to the variability of possible application scenarios and corresponding requirements, which include not only high estimation accuracy but also concern non-functional requirements, such as low power consumption of sensors.
Table 3.1 summarizes some of the most relevant recent studies in this field. Some works have the objective of classifying the input in order to predict the presence or absence of a vehicle. In this case, the performance is measured in terms of accuracy, sensitivity, number of false positives, etc.
Other works aim to estimate the number of vehicles (possibly of a particular class) that pass through the monitored structure in a given time interval. In this case, it is a regression task where performance is measured in terms of the error between the predicted value and the true value, using metrics as mean absolute error (MAE), often reporting this error as a percentage respect to the true value.
In the next paragraphs, for greater clarity, four of the works listed in the Table 3.1 will be described in detail.

Table 3.1: Recent works on TLE, categorized in terms of input data type, deployment scenario, algorithms, and results. Abbreviations: ANN: Artificial Neural Network, SVM: Support Vector Machine, V. vehicles, MAE: Mean Absolute Erros, Sens.: sensitivity, FP: False Positives

Work	**Data Type**	**Location**	**Algorithm**	**Main Result**
Kamkar et al. [2], 2016	Images	Bridge	Active Basis Model, Random Forest	Sens.: 74.82-92.11%, FP: 0.004-0.3
Dong et al. [3], 2018	Magnetic field	Freeway Exit	Classification Tree	Acc.: 99.8% (Single class) Acc.: 80.5% (Multi class)
Chen et al. [20]	Vibration	City	multi-step classifier	Acc.: 89.36% (Heavy V.) Acc.: 89.41% (Light V.)
Odat et al. [4], 2017	Ultrasound Infrared	City	Bayesian Networks	Sens.: 99%
Wang et al. [21], 2017	Magnetic field	City	Adaptive Threshold	Sens.: 97.5%
Liu et al. [22], 2019	Sound	City	Wavelet Denoising, Thresholding, SVM	MAE: 9.52-22.44%, Acc.: 71%
Ye et al. [23], 2020	Acceleration	City	Cross Correlation, ANN, K-means	Sens.: 87.5%

3.2.1 Vehicle detection, counting and classification in various conditions

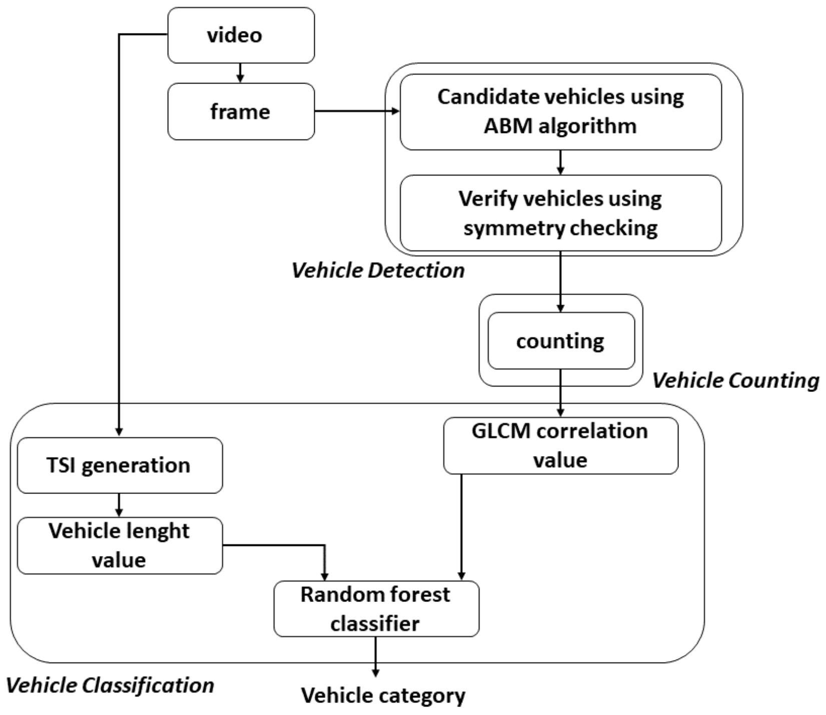


Figure 3.2: Method flowchart of [2]

In [2], the authors proposed a method to perform vehicle detection, classification, and counting on a public bridge (Figure 3.2). For the detection task, authors employ an Active Basis Model (ABM) [24] to process images provided by a smart camera in order to identify vehicle candidates. Once a vehicle candidate has been identified, it is verified through a symmetry evaluation algorithms that study the symmetry of the object detected dividing it into two parts according to the vertical axes from the middle and calculating the correlation between the left part and the mirror image of the right. The algorithm accept candidates whose symmetry is higher than a threshold. For vehicle counting task, authors set a single counting line in the middle of the frame where vehicles are expected to be detected. Vehicles passing this line are counted and are given to the classification section to be classified into one of the three possible categories: small (e.g. car), medium (e.g. van) and large (e.g. bus and truck). To perform the classification, authors extract the vehicle length in the time-spatial image (TSI) and also correlation of grey-level co-occurrence matrix (GLCM) obtained from the vehicle's bounding box. The extracted features are used to train a Random Forest (RF) classifier for categorisation. The RF grows many classification trees. To classify a new vehicle from its features, put the features down each of the trees in the forest which vote for one of the three possible classes. Finally the class that received the highest number of votes is chosen. The proposed algorithm can reach very high performance in high traffic situations up to 92.11% but is view-dependent, in fact the performance are strongly influenced by the light conditions of the bridge, decreasing to as much as 74.82%. The strong decrease of the performance in low-light conditions is one of the most important limitations of the camera sensors.

3.2.2 Improved Robust Vehicle Detection and Identification Based on Single Magnetic Sensor

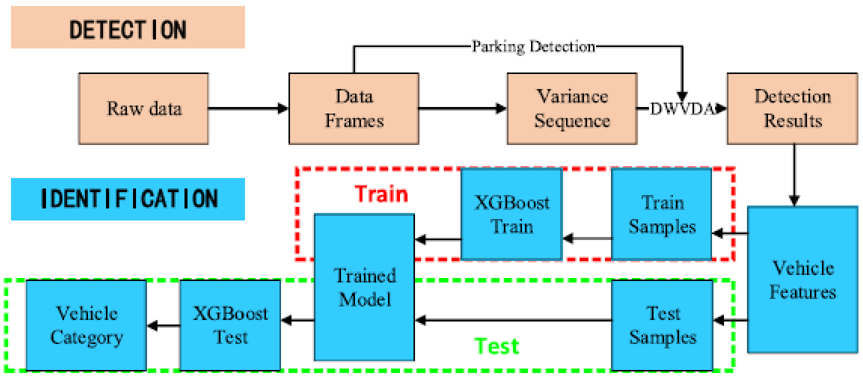


Figure 3.3: The Framework of the algorithm proposed in [3].

In [3] the authors proposed a vehicle detection algorithm that analyzes the signal provided by a single magnetic sensor (AMR sensor). Authors developed two different algorithms showed in Figure 3.3, one for vehicle detection, and the other for vehicle identification and categorization.

Vehicle Detection Algorithm

In the first step of the proposed algorithm, authors process the signal of the AMR sensor, converting the raw signal into a new signal that better represent the short-term variance. First the raw signal is segmented into short fragments called frames, then the variance sequence is obtained through calculating the variance of the data within every frame. Finally the signal is filtered through a smoothing filter, which takes a running average of the signal, removing the background random noise which is undesirable for the vehicle detection algorithm. The new signal is processed by a double window detection algorithm that is used to find the position of vehicles' arrival and leaving due to the difference between vehicle variance signal and background. Authors define two signal states in the detection: True state which represent a signal derived from vehicle disturbance and False state that represent a background signal. Testing this algorithm, authors reach an accuracy of 99,6%. The variance based detection algorithm can fail if vehicles are not in normal driving state, for instance a vehicle can be stopped above the sensor for a while. In order to manage this situation authors proposed a parking-sensitive improvement detection algorithm that recognize the magnetic intensity due to a vehicle stopped above the sensor, which is different from the intensity of background, introducing data frames into the detection algorithm to obtain a sliding background mean of data frames when there is no vehicle. Authors, comparing the sliding background mean and current magnetic strength mean, can determinate if the vehicle is stopped or it has passed the sensor. The parking-sensitive improvement has made the double window detection algorithm more robust increasing the accuracy to 99,8%.

Vehicle Identification Algorithm

Authors proposed a vehicle identification algorithm in order to classify the vehicles into four categories:

- Class 1: sedan and SUV
- Class 2: van and seven-seat
- Class 3: light and medium trucks
- Class 4: heavy truck and semi-trailer

To identify the type of a vehicle, authors employ the XGBoost [20], an advanced implementation of gradient tree boosting algorithm, training it with a large set of signal features. Authors extracted 42-D features are from every vehicle signal comprising statistical features of whole waveform and short-term features of fragment signal. In this task authors reach different accuracy for the four classes of vehicles. The best accuracy was achieved for class 1, with a score of 93,14% followed by class 4 (82,54%), class 3 (66,57%) and finally class 2 that with a score of 50,13%.

3.2.3 Vibration-Based Vehicle Classification System using Distributed Optical Sensing Technology

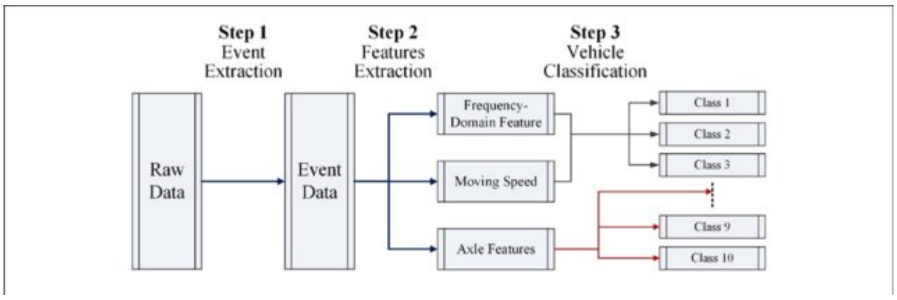


Figure 3.4: Classification procedures of [20].

In [20] authors develop a vibration-based vehicle classification system (Figure 3.4, that process signals provided by distributed optical vibration sensing (DOVS) technology and designed a multi-step classifier to categorize vehicles into 10 different classes based on vehicle type and number of axles. The system collect traffic-induced vibration signals, through an embedded sensing fiber as a distributed sensor and then authors extract several features from the raw signals to estimate axle configurations and identify vehicle categories. Authors process the raw signal with the empirical mode decomposition (EMD) in order to eliminate noises and extract useful signals. The EMD is a decomposition method that decompose raw signal into several intrinsic components, intrinsic mode functions (IMFs), which represents the detail components of the raw signal of different frequency bands. Authors used a short-time energy method to split the reconstructed signal into a series of short segments. Note that while the signal is non-stationary overall, dividing it into small segments results in a nearly-stationary signal. After the signal processing, authors extracted various features from the signal. The first step is the Event Extraction. In this step every short time energy are analyzed to determinate if a vehicle is present or not. Energies of time segments corresponding to a passage of a vehicle were significantly higher compared with the energies in "no vehicle". Then segments of input time-series corresponding to "vehicle event" and "no vehicle event" are extracted. After the Event Extraction authors used the time series extracted to extract Axle Features (axle number and axle space), Speed Estimation and Frequency-Domain Features. The last step consist in the vehicle classification based on the features extracted. Heavy vehicles (multi-axle trucks and trailers) are classified based on the axle configurations (number of axles and axle spacing). The vehicles that possess similar axle features (two-Axle Vehicles), are classified through a Support Vector Machine. To test and verify the system, a prototype system was installed on a relief road in Shanghai. The test shows good performance, the overall accuracy in the classification of heavy vehicles (number of axles is greater than three) was 89,36% and the overall accuracy in the classification of light vehicles was 89,41%.

3.2.4 Zero-Cost Hardware Vehicle Traffic Estimation in Structural Health Monitoring

The work described in [15] [16], present the idea of re-using SHM acceleration sensors to estimate the traffic load. In fact the goal of this work was to analyze the traffic passing through a single direction of travel through monitoring and processing vibrations collected from the SHM system described in section 3.1. This approach does not require any new installation or external source, while using only collected data from the inertial monitoring. Traffic monitoring means a series of surveys on the data collected for the purpose of detecting the passage of vehicles on the bridge in quantitative and qualitative manner. In this case, a quantitative analysis is used to distinguish normal time windows (in the absence of traffic), from anomalous temporal windows (presumably due to the passage of vehicles), thus associating the anomalies of the acceleration signals to the stresses due to vehicle traffic. The qualitative analysis finally has the goal to distinguish light vehicles from heavy vehicles. The Authors in order to train and test the algorithm that identify the passing of a vehicle integrated the unsupervised dataset, acceleration provided by the SHM system, with a new, smaller, supervised segment. They collected 1.20 hours of data and using a camera they synchronized the acceleration data of the sensors, X, with the labels, Y , obtained from video. They computed $y \in Y$ at a frequency of **100 Hz**, y can be either 0, 1, or 2, to indicate no vehicles, a car, or a truck, respectively.

Starting from the labelled dataset, the authors used a part of the dataset to train the hyper-parameters of their chain. In order to improve the signal-to-noise (SNR) ratio of the signal they use a wavelets-based de-noising algorithm, they exploit the wavelets to reduce the damping effects and the acceleration environmental noise on the signal to better distinguish between two near crossing vehicles. In particular, they applied an hard threshold on the coefficients of the wavelet algorithm, removing less informative components and maintaining only the most significant information. This process leads a better identification of vehicle peaks in following pipeline steps. The following step of the proposed pipeline consist in the application of a peak finding algorithm together with a sensor fusion approach in order to identify the vehicles crossing on the viaduct. A vehicle that cross on the viaduct causes a vibration at its natural frequencies which is detected from the accelerometers in the time domain as an amplitude peak. To analyze these peaks, authors decided to employ a peaks detection algorithm composed by two phases. In the first phase, authors identified two sets of feasible peaks, one that describes the cross of a light vehicle and an other that describes the cross of a heavy vehicle. Using this sets they applied two thresholds th1 (lower threshold) and th2 (highest threshold) to the preprocessed acceleration. All the time samples which exceed th2 are considered as candidates to be heavy vehicles, while all the time samples between th1 and th2 are considered as candidates to be light vehicles. The second phase of the peaks detection algorithm consist in the application of a refractory trainable period for each peak. The rationale for this second step resides in the dumping of the vibration observed on the viaduct. Authors experimentally found that inserting a trainable refractory period is beneficial to further filter out multiple peaks coming from the same vehicle. In the last part of the pipeline, they do a brief analysis of vehicle data demonstrating how to compute vehicle speed in low traffic condition.

Chapter 4

Methods And Algorithms

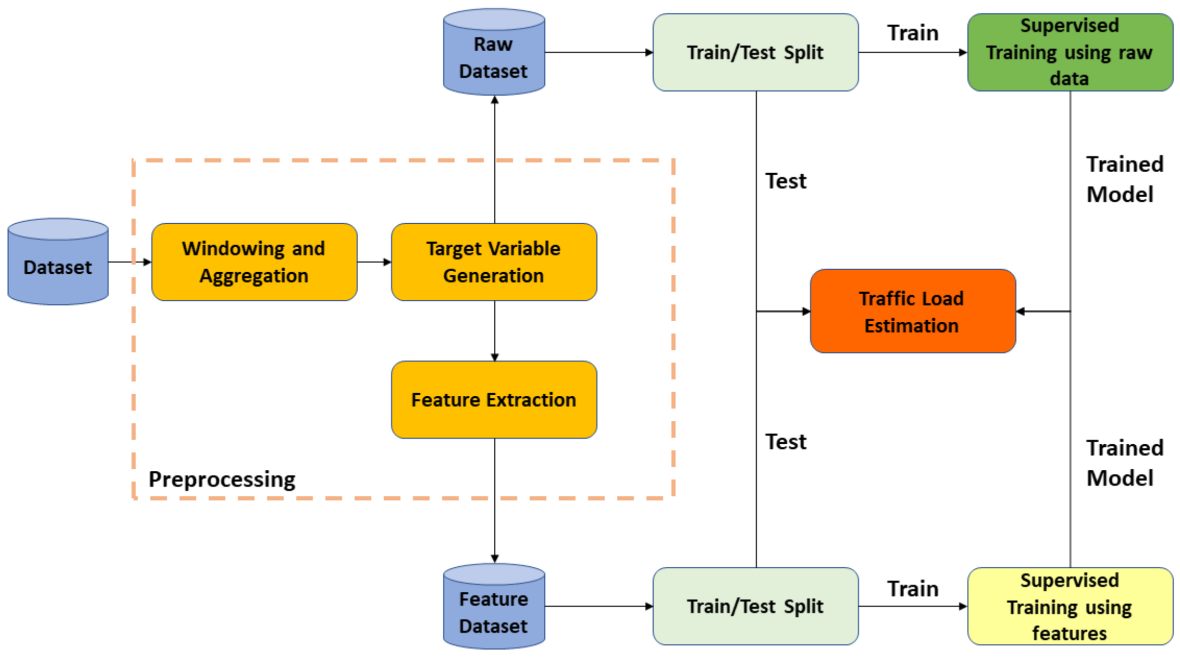


Figure 4.1: Proposed pipeline for supervised learning-based traffic load estimation using SHM sensors.

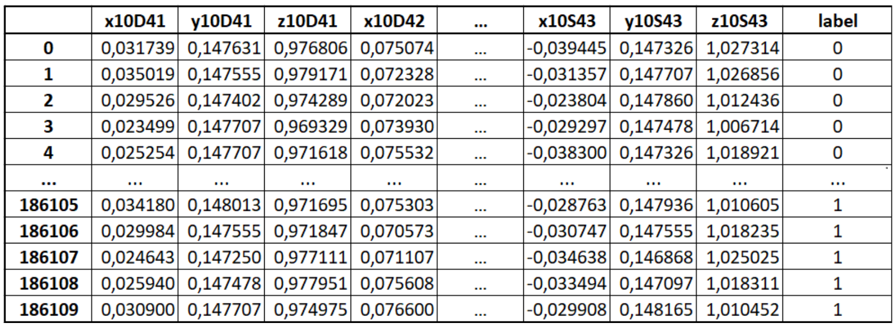
Figure 4.1 shows the supervised learning pipeline used in this study. The novelty of this approach does not lies in any of the individual phases of the pipeline, which are standard for similar tasks [25][26], but in applying them

for the first time to the problem of traffic load estimation based on SHM sensors.
In this chapter, the techniques and algorithms used for the implementation of each step of the pipeline in Figure 4.1 will be explained in detail. The purpose of this Pipeline is to train various ML and DL algorithms in a supervised manner with the aim of estimating the number of vehicles passing through the viaduct in various time windows.
Precisely, the regressors were trained to separately estimate the count of light vehicles (e.g. cars) and heavy vehicles (e.g. trucks). As we can see from the figure, a preprocessing phase is applied to the starting dataset (described in section 4.1.1).
Section 4.1 describes in detail the preprocessing techniques (orange boxes in the figure) used, which allow us to obtain two new datasets for each time interval examined.
The first dataset, "Raw Dataset" in the figure, contains the accelerations present in the starting dataset aggregated in time windows, and for each window the corresponding labels, y_{light} and y_{heavy}, which respectively represent the number of light and heavy vehicles that crossed the viaduct in that window.
The second dataset, "feature dataset" in the figure, is generated by the feature extraction phase (described in section 4.1.3) and contains the features extracted from the "Raw Dataset" and the corresponding labels.
Section 4.2 describes in detail the ML and DL algorithms trained to perform the regression. We can see in the figure that the two new datasets are used to train two different groups of algorithms.
The "Raw Dataset" is used to train, (green box in Figure 4.1), various Deep Neural Network models (described in sections: 4.2.6 and 4.2.7), which as we have seen in section 2.4 internally implement the feature extraction phase.
The "Feature Dataset" is used to train (yellow box in Figure 4.1), various traditional ML networks (described in sections: 4.2.1, 4.2.2, 4.2.3, 4.2.4) and MLP models(described in sections: 4.2.5).

4.1 Data Preprocessing

4.1.1 Starting Dataset

The Starting Labeled Dataset (Figure 4.2) used in this study was produced in the work described in [15][16].



	x10D41	y10D41	z10D41	x10D42	...	x10S43	y10S43	z10S43	label
0	0,031739	0,147631	0,976806	0,075074	...	-0,039445	0,147326	1,027314	0
1	0,035019	0,147555	0,979171	0,072328	...	-0,031357	0,147707	1,026856	0
2	0,029526	0,147402	0,974289	0,072023	...	-0,023804	0,147860	1,012436	0
3	0,023499	0,147707	0,969329	0,073930	...	-0,029297	0,147478	1,006714	0
4	0,025254	0,147707	0,971618	0,075532	...	-0,038300	0,147326	1,018921	0
...	...	...	...	...	...	...	...	...	...
186105	0,034180	0,148013	0,971695	0,075303	...	-0,028763	0,147936	1,010605	1
186106	0,029984	0,147555	0,971847	0,070573	...	-0,030747	0,147555	1,018235	1
186107	0,024643	0,147250	0,977111	0,071107	...	-0,034638	0,146868	1,025025	1
186108	0,025940	0,147478	0,977951	0,075608	...	-0,033494	0,147097	1,018311	1
186109	0,030900	0,147707	0,974975	0,076600	...	-0,029908	0,148165	1,010452	1

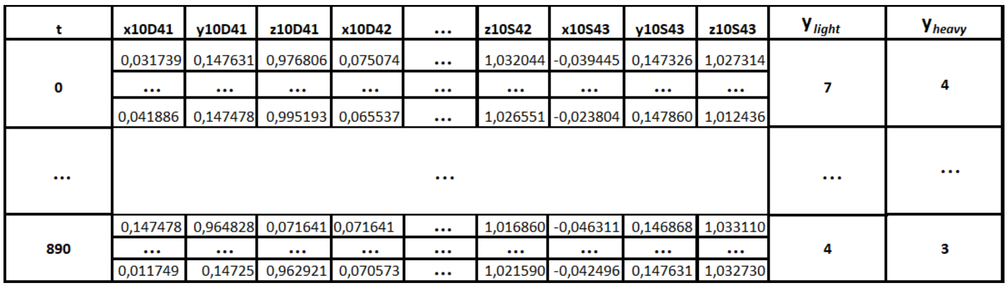
Figure 4.2: Starting Labeled Dataset. Columns [(x/y/z)10(D/S)4x] represent the acceleration (m/s^2) detected by the 4x sensor in the specific axis (x, y or z).

The Dataset consists of 31 minutes of acceleration data collected by the 7 accelerometers placed in section 10 of the viaduct, where the system SHM (described in section 3.1) is located. The dataset reports, for each sensor, the accelerations in the three axes: "x" axis, "y" axis and "z" axis. So there are $7 * 3 = 21$ columns of acceleration data.
The accelerations collected by the sensors were sampled at a frequency of 100 Hz, so the dataset is composed of $31 * 60 * 100 + 1{,}1 * 100 = 186110$ acceleration rows. As we can see in the Figure 4.2, in the last column for each sample there is a label.
The label was assigned by examining a video, recorded in synchrony with the collected accelerations, of the road portion corresponding to section 10 of the viaduct. The video, recorded at 10 FPS, was processed with an object detection algorithm, to have an initial estimate of the presence of vehicles. A human operator then went through the entire video to manually fix the error detection errors.
The label describes if there are vehicles crossing the section under consideration in the corresponding time.
The values of the label can be:

- 0 no vehicle;
- 1 light vehicle;
- 2 heavy vehicle;

Since the frame rate of the video and the acceleration sampling frequency are in a 1:10 relation (10 fps vs 100Hz), slices of 10 consecutive acceleration samples received the same label. So the presence of ten consecutive "1" indicates the passage of a light vehicle, while the presence of ten consecutive "2" indicates the passage of a heavy vehicle.

4.1.2 Windowing and Target Variable Generation



t	x10D41	y10D41	z10D41	x10D42	...	z10S42	x10S43	y10S43	z10S43	y_{light}	y_{heavy}
0	0,031739	0,147631	0,976806	0,075074	...	1,032044	-0,039445	0,147326	1,027314	7	4
	...	...	...	...	...	...	...	...	...		
	0,041886	0,147478	0,995193	0,065537	...	1,026551	-0,023804	0,147860	1,012436		
...	...									...	...
890	0,147478	0,964828	0,071641	0,071641	...	1,016860	-0,046311	0,146868	1,033110	4	3
	...	...	...	...	...	...	...	...	...		
	0,011749	0,14725	0,962921	0,070573	...	1,021590	-0,042496	0,147631	1,032730		

Figure 4.3: Example of a Raw Dataset created by the windowing and aggregation and the target variable generation pipeline's step.

This section describe the two steps of the pipeline (Figure 4.1) that product the "Raw Datasets" namely the windowing and aggregation step and the target variable generation step.
Since the goal of this thesis is to test various algorithms capable of calculating the number of light vehicles and heavy vehicles passed in a given time window, the first step was to create new Datasets that group the starting data in different time windows.
In particular, the new Datasets were created using a windowing function that groups the Dataset into windows of 5s, 6s, 7s, 8s, 20s, 30s, 40s, 50s, 60s respectively, applying a shift of 2 s.
The new Datasets contain 24 columns and 891 blocks with different number of rows based on the windows size. As we can see in Figure 4.3, The first column represents the block identification number, from 0 to 890, the following 21 columns contain the acceleration values (in the three axes) of the 7 accelerometers taken into consideration.
Finally the last two columns contain respectively the y_{light} and y_{heavy} labels identifying the number of light and heavy vehicles passed in the corresponding time window. The number of rows of each block depends from the window size. The number of samples contained in windows of 5s, 6s, 7s, 8s, 20s, 30s, 40s, 50s, 60s are respectively 500, 600, 700, 800, 2000, 3000, 4000, 5000, 6000. The y_{light} and y_{heavy} labels are calculated in the target variable generation

step of the pipeline, applying for each window the following equations:

$$y_{light} = \frac{\sum_{t=1}^{T}(l_t = 1)}{10} \tag{4.1}$$

$$y_{heavy} = \frac{\sum_{t=1}^{T}(l_t = 2)}{10} \tag{4.2}$$

Notice that the two new labels can also be a fractional number due to the shift for which the series of ten "1" or ten "2" can be split into 2 consecutive blocks.

4.1.3 Feature Extraction

This section describes the last step of the data preprocessing phase, of the pipeline in Figure 4.1, i.e. feature extraction.
For each window, 12 independent statistical features relative to the accelerometer axis of each of the sensors have been extracted. These are well known and commonly used for other supervised machine learning tasks dealing with acceleration data. After feature extraction, each sliding window reduces to a vector of $n = 21 * 12 = 252$ variables.
These features were extracted using methods from the numpy [27] and scipy stats [28] libraries.
The remainder of this section will give a brief description of each extracted feature and the method used to extract it.

- **Mean acceleration**:

This feature simply represents the arithmetic mean along the specified axis. To extract this features the numpy.mean() function was used which applies to each column of the window the following function:

$$mean = \frac{\sum_{i=0}^{N-1}(x_i)}{N} \tag{4.3}$$

- **Acceleration standard deviation:**

This feature represents the standard deviation along the specified axis. The standard deviation (σ) is a measure of the amount of variation or dispersion of a set of values [29]. A low value of σ indicates that the values of the examined set, tend to be close to the mean (also called the expected value), in contrast a high value of σ indicates that the values are spread over a wide range. To extract these features the numpy.std() function was used which applies to each column of the window the following function:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(x_i - \mu)^2} \tag{4.4}$$

- **Minimum acceleration sample:**

This feature simply represents the minimum value along the specified axis. To extract this features the numpy.min() function was used which extract the lowest value to each column of the window.

- **Maximum acceleration sample:**

This feature simply represents the maximum value along the specified axis. To extract this features the numpy.max() function was used which extract the highest value to each column of the window.

- **Acceleration median**

This feature represents the median value along the specified axis.
In statistics and probability theory, the median is the value that separates the upper half from the lower half of a data sample.
The key feature of the median is that, unlike the mean, it is not strongly affected by a small portion of extremely high or low values, and thus providing a better representation of a "typical" value [30]. To extract this features the numpy.median() function was used which extract to each column of the window the following value:

$$Median = X_{sorted}[(N-1)/2] \tag{4.5}$$

Given an array X of N elements, the Median function first sorts the array and then extract the central value in position (N-1)/2.

- **Kurtosis coefficient:**

This feature represents the Fisher's coefficient of kurtosis value along the specified axis.

In statistics, the kurtosis coefficient is used to describe the shape of a probability distribution with respect to the normal (Gaussian) distribution [31].
The data sets with a high value of kurtosis are called heavy-tailed, that is, unlike the normal distribution they tend to have a high number of values at the extremes of the distribution curve or outliers. The extreme case would be a uniform distribution [32].
Data sets with low kurtosis value are called light-tailed, i.e. they tend to have a low number of values at the extremes of the distribution curve or lack of outliers. The value corresponding to a Gaussian distribution is 0.
To extract this features the scipy.stats.kurtosis() function was used which applies to each column of the window the following function [33]:

$$Kurtosis = \frac{n^2 * (n+1) * M_4}{(n-1) * (n-2)(n-3) * \sigma^4} - 3 * \frac{(n-1)^2}{(n-2) * (n-3)} \tag{4.6}$$

where:

$$M_4 = \frac{\sum_{i=1}^{N}((X_i - \bar{X})^4)}{n} \tag{4.7}$$

- **Skewness index:**

This feature represents the sample skewness along the specified axis. The skewness is a measure of the symmetry of the shape of a probability distribution. A distribution, or dataset, is symmetric if it presents the same shape in the left and in the right compared to the center point. The skewness value for a normal distribution is zero. Negative skewness values indicate that the distribution of the data is greater to the left than the mean while positive values indicate that the distribution is greater to the right of the mean [34].
To extract this features the scipy.stats.skew() function was used which compute to each column of the window the Fisher-Pearson coefficient of skewness:

$$Skew = \frac{m_3}{m_2^{3/2}} \tag{4.8}$$

where:

$$m_i = \frac{1}{N}\sum_{n=1}^{N}(x[n] - \bar{x})^i \tag{4.9}$$

- **Root Mean Square:**

This feature simply represents the Root Mean Square (RMS) along the specified axis. RMS is also known as quadratic mean [35][36] and is a special case of generalized mean with exponent 2. RMS is a statistical dispersion index, which is an estimate of the variability of a data population or a random variable. To extract this features the numpy.sqrt(numpy.mean(X^2)) function was used which applies to each column (X) of the window the following mathematical function:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i^2)} \tag{4.10}$$

- **Sum of the absolute values:**

This feature simply represents the sum of the absolute values along the specified axis. To extract this features the numpy.sum(numpy.abs(X)) function was used which applies to each column (X) of the window the following mathematical function:

$$sumabs = \sum_{i=1}^{N}(|x_i|) \tag{4.11}$$

- **Element over mean:**

This feature simply represents the count of the elements larger than the mean along the specified axis.

- **Acceleration Energy:**

This features represent the sum of the squared elements along the specified axis. The Acceleration Energy is defined mathematically as:

$$Energy = \sum_{i=1}^{N}(x_i^2) \tag{4.12}$$

- **Median Absolute Deviation:**

This features represent the median over the absolute deviations from the median of the elements along the specified axis. It is a measure of dispersion similar to the standard deviation but more robust to outliers [37]. To extract this features the scipy.stats.median_abs_deviation() function was used which compute to each column of the window the following function:

$$MAD = median(|X_i - \tilde{X}|) \tag{4.13}$$

where is the median of X (see equation 4.5)

4.2 Regression Algorithms

This section details ML and DL regressors trained and tested to perform SHM sensor-based TLE. For each type of regressor a brief theoretical description of its functioning will be given and the hyperparameters explored and chosen will be described. As a general consideration valid for all algorithms, better results are obtained by setting two separate regressors to estimate y_{light} and y_{heavy}, i.e. the traffic load of light and heavy vehicles, respectively, compared to training a single model to produce an output vector (light; heavy). Therefore, we trained all models listed below twice and independently as scalar regressors, once for each target output. The choice of hyperparameters was made by testing them on the dataset composed of 60s windows, which proved to be the one that allows to obtain the best results.

4.2.1 Linear Regressor (LR)

The simplest algorithm that has been considered as a candidate for the TLE is a linear regressor, which also serves as a baseline for more complex models. In statistics, there are three types of linear regression:

- **Simple linear regression:** this is the case in which there is one independent variable (feature) and one dependent variable (target).
- **Multiple linear regression:** this is the case in which there are n independent variables (features), where $n \geq 2$, and one dependent variable.
- **Multivariate linear regression:** this is the case in which there are n_1 independent variables (features) and n_2 dependent variables (targets) with $n_1, n_2 \geq 2$.

The linear regressor that is trained in this thesis performs a multy variable regression, as it is trained to predict the discrete value of lebels **($\mathbf{y}_{light}$ or $\mathbf{y}_{heavy}$)** starting from a vector of **212 independent statistical features**. Precisely, the Linear Regression Model from the scikit learn (sklearn) library was used using the default hyperparameters [38].
The internal parameters of the regressor are modified during the train phase in order to minimize the sum of the squared errors between predicted and observed targets.

A linear regression model assumes that the relationship between the dependent variable y and the vector of the features x is linear, therefore the mathematical function which aims to represent the relationship between the independent variables (the features x) and the independent one (the y to predict) has the following form:

$$y = B_0 + x_1 * B_1 + x_2 * B_2 + ... + x_n * B_n + \epsilon \tag{4.14}$$

Where y is the dependent variable to be predicted. The x_i are the independent variables (in our case the 212 features extracted for each window), the B_i are the internal coefficients and ϵ represents the residual error, a random variable which adds "noise" to the linear relationship between the dependent variable and the regressor.

4.2.2 Decision Tree (DT)

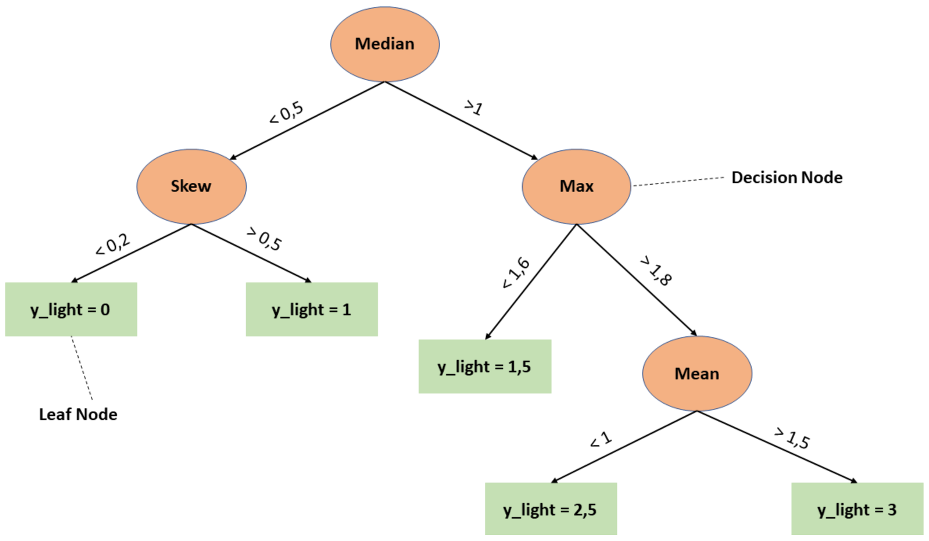


Figure 4.4: Example of a portion of a decision tree applied to the TLE .

The Decision Tree (DT) is a supervised machine learning model that builds regression models by developing decision rules starting from the features, in the form of a tree structure (see Figure 4.4). The DT breaks down a data set into smaller and smaller subsets by incrementally developing an associated decision tree. The final result is a tree with decision nodes and leaf nodes.

A decision node (e.g. max in Figure 4.4) has two or more branches, each representing values for the tested attribute. The leaf node (e.g. $y_{light} = 0$ in Figure 4.4) represents a predicted discrete value. The top decision node in a tree, that matches the best predictor attribute, is called the root node.
In this thesis the DT from the scikit learn library [38] was trained using the features dataset, described in section 4.1.3 to perform the TLE. The main hyperparameters of this model are:

- **Criterion{"mse", "friedman$_{mse}$", "mae", "poisson"}:**

Represents the mathematical function used to measure the quality of a division carried out by the decision nodes. The supported criterion are:
Mean Squared Error (mse) which minimizes the Least Square Errors (L2 loss equation 4.15) by using the average of each terminal node.

$$L2_{loss} = \sum_{i=1}^{N} (y_{true} - y_{predicted})^2 \tag{4.15}$$

Mean absolute error (mae) which minimizes the Least Absolute Deviations (L1 loss equation 4.16) using the median of each terminal node

$$L1_{loss} = \sum_{i=1}^{N} |y_{true} - y_{predicted}| \tag{4.16}$$

Friedman$_{mse}$ which uses the mean square error with Friedman's improvement score for the potential divisions.
Poisson which uses Poisson's deviance reduction (equation 4.17) to find divisions.

$$D = 2\sum_{i=1}^{n} \{Y_i \log(Y_i/\mu_i) - (Y_i - \mu_i)\} \tag{4.17}$$

where if $Y_i = 0$, the $Y_i \log(Y_i/\mu_i)$ term is taken to be zero, and μ_i (equation 4.18) denotes the predicted mean for observation i based on the estimated model parameters [39].

$$\mu_i = \exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + ... + \hat{\beta}_p X_p) \tag{4.18}$$

In the model trained in this work, the mse criterion was selected.

- **splitter{"best", "random"}:**

Represent the strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose a random split. The strategy chosen in our model is "best".

- **max_depth {int}:**

This hyperparameter limits the maximum depth of the tree to a integer number. Different depth values [10, 50, 100, 200, 300, 400] have been considered for our model, was selected max_depth = 200 which obtained the best performance minimizing the validation error.

4.2.3 K-Nearest Neighbors (KNN)

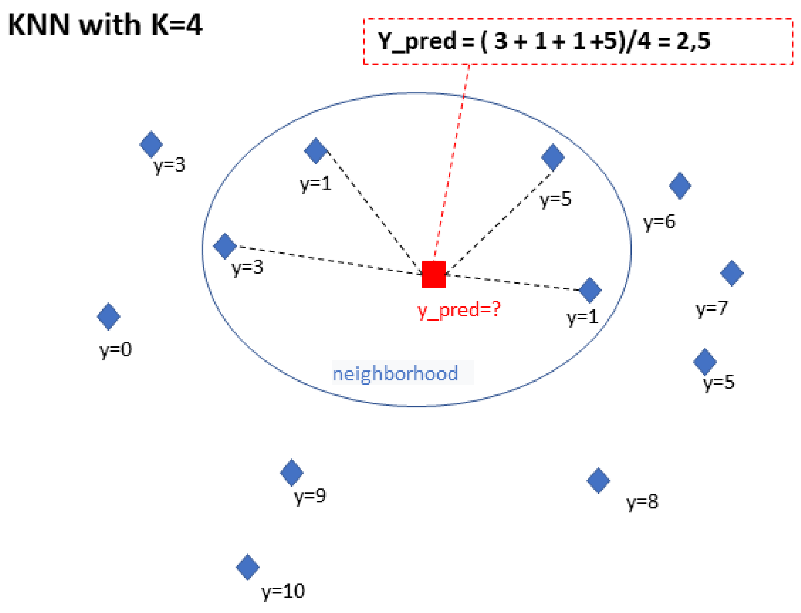


Figure 4.5: Example of a KNN regressor with k=4.

The K-Nearest Neighbors (KNN) regression is a non-parametric method, based on the features of the near instances [40]. KNN approximates the association between independent variables (features) and the dependent variable (y) by calculating the mean of the k closest observations.
In this thesis the KNN from the scikit learn library [38] was trained using the features dataset, described in section 4.1.3 to perform the TLE.

The main hyperparameter of this algorithm is the k value, which consists of the number of neighbors to consider in predicting the target. In fact, the KNN regressor splits the observations into groups called neighborhoods, populated by k samples.
In the choice of this hyperparameter we tested the performance of the regressor with the values of $k \in [1, 3, 5, 7, 9, 11]$ obtaining the best results for $k = 7$.
KNN selects the k samples to be considered as neighbors, through a function that estimates the similarity between the features of the various samples. Since in our case all the input features are numerical, we measured the similarity among the samples as the inverse of the Euclidean distance. The k elements that have a smaller value of Euclidean distance between their features are considered to belong to the same neighborhood.
Mathematically the choice of each neighbor is described in equation 4.19.

$$neighbor_i = min\{d(n_1, x), d(n_2, x), \ldots, d(n_i, x)\} \tag{4.19}$$

where n_i are the potential neighbors, x is the sample for which we want to predict the target and the function d is the Euclidean distance equation 4.20.

$$d = \sqrt{\sum_{i=1}^{n} (n_{feature(i)} \breve{\ } x_{feature(i)})^2} \tag{4.20}$$

Once the k neighbors have been selected, the regressor estimates the target value to be predicted as the arithmetic mean of the y values of the k elements belonging to the neighborhood (equation 4.21).

$$y_{pred} = \frac{\sum_{i=1}^{k}(y_i)}{k} \tag{4.21}$$

For greater clarity, the Figure 4.5 shows an example of prediction through a KNN regressor with $k = 4$.

4.2.4 Support Vector Regressor (SVR)

The last classic ML algorithm employed is the Support Vector Regression (SVR), which is the equivalent of a Support Vector Machine (SVM) for regression problems.
SVR is a specialization of Support Vector Machine (SVM) which sees its major applications in classification problems. To understand the operation of the

SVR we start from the principles of the SVM, describing its functioning in the solution of classification problems. SVM are supervised discriminant machine learning techniques. Unlike generative automatic learning approaches, which require calculations of conditional probability distributions in the prediction phase, in discriminant learning techniques a discriminating function is found in the training phase. This function is capable of labeling new observations in one of the possible classes. In the case of the SVM, this function is represented by the hyperplane that separates the samples in space. The

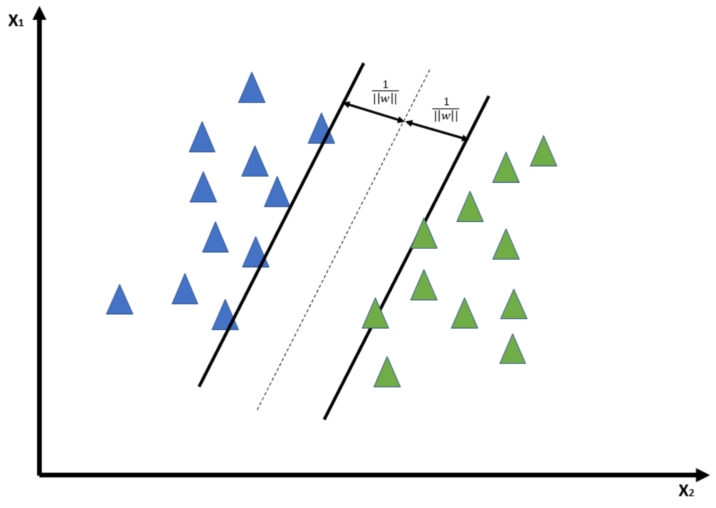


Figure 4.6: Hard margin hyperplane

simplest case is the binary classification, in which we have two classes. In the training phase, the SVM calculates the hyperplane (equation 4.22) that best separates the class samples.

$$g(x) = w^T x + b \tag{4.22}$$

This hyperplane found separates the two classes with their maximum distance. The goal is to find w and b such that the distance between the hyperplane and the nearest points x_i, of both classes, is maximized. We can see in Figure 4.6 that if the training data are linearly separable, the algorithm identifies two parallel hyperplanes that separate the two data classes, in order to have the maximum possible distance between them. The region bounded by these two hyperplanes is called the boundary and the hyperplane with maximum

margin is the hyperplane that is halfway between them.
Geometrically it means to have a margin of [41]:

$$\frac{1}{||w||} + \frac{1}{||w||} = \frac{2}{||w||} \tag{4.23}$$

whereas $w^T x + b = 1$ for $x \in class1$ and $w^T x + b = -1$ for $x \in class2$
This leads to an optimization problem that minimizes the objective function:

$$J(w) = \frac{1}{2}||w||^2 \tag{4.24}$$

subject to the constraint:

$$y_i(w_i^T x + b) \geq 1, i = 1,2, ..., N \tag{4.25}$$

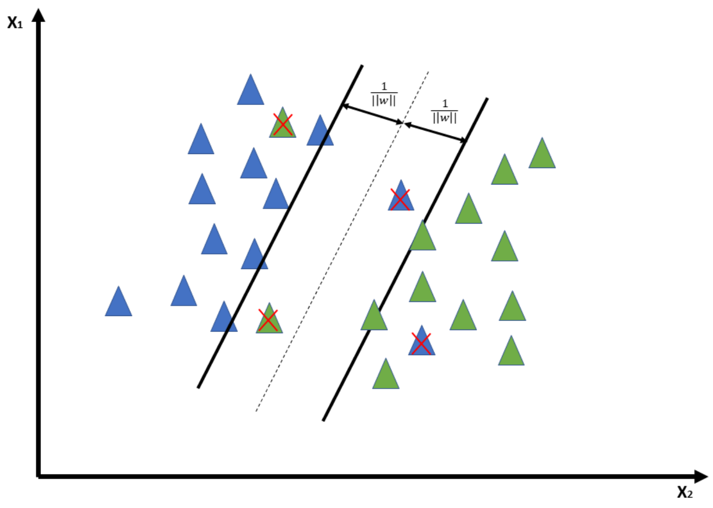


Figure 4.7: Soft margin hyperplane

When the data is not completely separable, you can choose to use the Soft Margin version of linear SVM (Figure 4.7), introducing the offset variables ξ_i in the objective function to allow errors in the classification.
The new objective function to minimize will be [41]:

$$J(w, b, \xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i \tag{4.26}$$

subject to these two constraints:

$$y_i[w_i^T x_i + b] \geq 1 - \xi_i, i = 1,2,...,N\xi_i \geq 0, i = 1,2,...,N \tag{4.27}$$

SVM, in this case, is not looking for the hard margin, which will classify all the data impeccably but agrees to have a soft margin by correctly classifying most of the data and allowing the model to misclassify some points near the separation boundary .
Often, the data is not linearly separable in the original input space. Instances that have different labels share the input space in a way that prevents a linear hyperplane from properly separating the different classes. If Soft margin SVM cannot find a fairly robust separation hyperplane, that is, one that has an acceptable number of incorrect classifications, SVM solves this problem by mapping the data in a multidimensional space through the use of predefined kernel functions. In this new space with more dimensions, a linear separator could be able to discriminate between the different classes. There are various kernel functions, the most used are:

- **Linear kernel:**

$$k(x,y) = x^T y + c \tag{4.28}$$

- **Polynomial function:**

$$k(x,y) = (\alpha x^T y + c)^d \tag{4.29}$$

- **Gaussian radial basis function (RBF):**

$$k(x,y) = exp\left(-\frac{||x-y||^2}{2\sigma^2}\right) \tag{4.30}$$

- **Sigmoid kernel:**

$$k(x,y) = tanh(\alpha x^T y + c) \tag{4.31}$$

The regression problem is a generalization of the classification problem, in which the model returns a continuous-valued output, rather than an output from a finite set. Support Vector Regression (SVR) uses the same principles of SVM for classification, with the difference that it is necessary to introduce a tolerance margin ϵ that allows an approximation of the continuous value, which otherwise would have infinite possibilities [42]. For the rest, the same considerations made for SVM apply, i.e. the goal is always to minimize the error, identifying the hyperplane that maximizes the margin, bearing in mind that part of the error is tolerated. Also in SVR you can use the same kernel functions seen above to map data in a multidimensional space in order to make it linearly separable. In this thesis the SVR from the scikit learn library [38] was trained using the features dataset, described in section 4.1.3 to perform the TLE. In the SVR configuration, the values of two fundamental hyperparameters were tested:

- **Hyperparameter C**, which indicates to the SVR optimization what weight you want to give to the incorrect classifications for each example of the training set. For high values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of correctly classifying all training points. In contrast, a very small value of C will cause the optimizer to look for a hyperplane that separates the larger margin, even if that hyperplane incorrectly ranks more points. Values 1 and 10 were tested for this hyperparameter.

- **The kernel function**, for which the linear kernel and the rbf kernel have been tested.

The configuration that gave the best results was the one with **C = 10** and **Gaussian radial basis function (RBF) kernel**.

4.2.5 Multy Layer Perceptron (MLP)

Multy Layer Perceptron (MLP) is the first Deep Learning architecture tested in this work. Multilayer perceptron (MLP) is the standard network type of feed forward architectures and has historically received great interest as it is an architecture suitable for solving various categories of problems. MLP is an evolution of the Preceptron model introduced in 1958 by Frank Rosenblatt [43] which was composed of a single neuron and could only perform binary classification tasks. As shown in 4.8, MLP is composed of three types of percetron layers: input layer, output layer and hidden layer.
The input layer receives the input signal to be processed. An arbitrary number of hidden layers that are placed between the input and output layers are the real computational engine of the MLP [44]. The required activity such as regression and classification is performed by the output layer. In the case of a regression task, the output layer is composed of a single neuron,

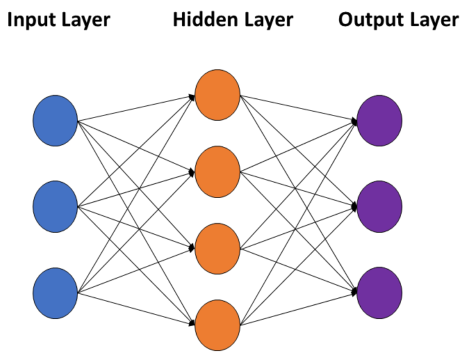


Figure 4.8: Multilayer Perceptron.

which will output the discrete value predicted by the network while in the case of a classification task the output layer is usually composed of n neurons, where n is the number of classes to predict.
The MLP architecture is called fully connected as each neuron of a layer is connected with a certain weight w_{ij} to each neuron of the next layer.
Neurons in the MLP are trained with the backward propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems that are not linearly separable. In this thesis the Multy Layer Perceptron regressor from the scikit learn library [**sklearnsvr**] was trained to perform the TLE.
As explained in section 2.3 the Deep Learning architectures, therefore also MLP, are able to internally extrapolate the features starting from the raw data, On the other hand, MLP is not optimal for the management of data series. Therefore, it was decided to train MLP using the features dataset (described in section 4.1.3) used to train the classic ML architectures seen above. All versions of MLP were trained using the Adam gradient-based optimizer, a learning rate of 10^{-3} and a mini-batch size of 200 samples.
The most important hyperparameter of this architecture is the size of the network, in fact MLP allows to set the number of hidden layers and the number of neurons that make up each layer.
In order to find the implementation that achieved the best performance, various settings of number of hidden layers and number of neurons were explored.

MLPs with 2 or 3 hidden layers composed of $n_{neurons} \in [10, 20, 50, 100, 200]$ were tested. The implementation that achieved the best results was the MLP consisting of **3 layers** of **100 neurons** each.
In each layer, the Mean Squared Error (MSE) was used as the loss function and the Rectified Linear Unit (ReLU) as the activation functions.

4.2.6 Recurrent Neurel Network Models

As we saw in section 2.4.1, Recurrent Neural Networks (RNNs) are a very powerful Deep Neural Network architecture for modeling sequence data as a time series. There are various variants of RNN. In this thesis we will explore three highly successful RNN variants as possible candidates for TLE based on SHM sensors.
This section describes the implementations of the three RNN variants, namely the standard Recurrent Neural Networks (SimpleRNN), the Long Short-Term Memory (LSTM) and the Gated Recurrent Units (GRU).
The proposed models were created using the keras sequential model [45], which allows to stack sequences of layers provided by the keras API [45].
Unlike the models previously described, the RNN models implemented in this thesis are not trained with the extracted features, but being able to internally extrapolate the features, as we can see in the green box in Figure 4.1, they are trained with the raw data. The raw dataset (described in section 4.1.2) is used in three different versions to train and test these models. These three versions differ according to the type of preprocessing applied, namely:

- The original version (no preprocessing).
- The standardized version (preprocessed by applying the StandardScaler).
- The normalized version (preprocessed by applying the MinMaxScaler in the range [0,1]).

Each model was trained for 100 training epochs using the Adam gradient-based optimizer, a learning rate of 10^{-3} and a mini-batch size of 128 samples.We used the Mean Squared Error (MSE) as a loss function and hyperbolic tangent (tanh) activation functions in all layers.

Standard Recurrent Neural Network (SimpleRNN)

The first implementation considered is the simplest, the SimpleRNN. Its operating principles are explained in section 2.4.1. The SimpleRNN model is composed as shown in the table 4.1.

Layer (type)	Param
simple_rnn (SimpleRNN)	5504
dropout (Dropout)	0
simple_rnn (SimpleRNN)	8256
dropout (Dropout)	0
dense (Dense)	65

Table 4.1: SRNN model
Total params: 13,825
Trainable params: 13,825

The key layers of this sequential model are the keras.layers.SimpleRNN, a fully-connected RNN where the output from previous timestep is to be fed to next timestep [45]. Each layer SimpleRNN is composed of 64 neurons.
The Dropout layer randomly sets input units to 0 with a frequency of (rate = 0,2) at each step during training time, which helps prevent overfitting.
The model output is calculated from a Dense Layer, similar to an MLP, that computes the dot product between the inputs (output of the last SimpleRNN layer) and the kernel along the last axis of the inputs and axis 1 of the kernel.

Long-Short Term Memory (LSTM)

Long Short Term Memory networks (LSTM) are a variant of RNN introduced by Hochreiter Schmidhuber (1997) [46], capable of learning long-term dependencies. All recurrent neural networks operate as a chain of repeating neural network modules. The modules that make up a LSTM, also placed as a chain, differ from those of the standard RNN for their greater complexity. The fundamental element of an LSTM module is the cell state, which operates as a long-term memory that can be adjusted by the network through three gates that regulate the flow of information in the cell. Each gate consists of a sigmoid activation, which regulates the flow of information multiplying the data by a value between 0 and 1.
The input gate has the task of managing the flow of input data to the cell quantifying the importance of each input data.
The forget gate manages the information present in the cell state, deciding

whether to keep it or forget it, if it deems it irrelevant, the data in the cell is multiplied by 0, therefore forgotten [47].
The output gate manages the output flow to other hidden layers, this gate too, as seen for the others, multiplies the data in transit by a value between 0 and 1 based on its relevance [48]. The proposed LSTM model is composed as shown in the table 4.2.

Layer (type)	Param
lstm (LSTM)	22016
dropout (Dropout)	0
lstm (LSTM)	33024
dropout (Dropout)	0
dense (Dense)	65

Table 4.2: LSTM model
Total params: 55,105
Trainable params: 55,105

The two key layers of this sequential model are keras.layers.LSTM [45], both of which are composed of 64 units working as described above.

Gated Recurrent Unit (GRU)

Gated recurrent unit (GRU) is an advanced variant of RNN introduced by Kyunghyun Cho et al in the year 2014 [49].
GRU is an architecture very similar to the Long Short Term Memory (LSTM) described above. In fact, both are based on the principle of managing the flow of information through the gates [50].
GRU implementation is much simpler than LSTM, in fact, unlike LSTM, GRU modules do not have a cell state, but only a hidden state like standard RNN. As mentioned above also GRU is based on the management of the data flow through the gates, but unlike LSTM each GRU module has two gates and not three.
The two gates of GRU are the reset gate and the update gate. They are two vectors that have the task of deciding which information to pass to the output. Their peculiarity is that they can be trained to remember information for a long time.

The update gate is responsible for the long term memory, helping the model to choose which information from past instants to transmit to future instants of time. As in the LSTM gates, this operation takes place by applying a sigmoid activation function that multiplies the information by a value between 0 and 1, where 0 means forgetting the information and 1 keeping it completely.
The reset gate is responsible for managing the information contained in the memory in the hiden state, using a sigmoid activation function as the update to forget or keep the information [51]. The proposed GRU model is composed as shown in the table 4.3. The two key layers of this sequential model are

Layer (type)	Param
gru (GRU)	16704
dropout (Dropout)	0
gru (GRU)	24960
dropout (Dropout)	0
dense (Dense)	65

Table 4.3: GRU model
Total params: 41,729
Trainable params: 41,729

keras.layers.GRU [45], both of which are composed of 64 units working as described above

4.2.7 Temporal Convolutional Network (TCN)

The latest architecture tested in this work is a Convolutional Neural Network (CNN). Since this is also a DL architecture able to internally extrapolate the features starting from the raw data, it was trained using the same datasets seen for the RNNs. CNNs are designed to work on multiple arrays of data, for this reason they are particularly useful in the treatment of images. In fact the color images are represented by three two-dimensional arrays containing the intensities of the pixels with respect to the three colors, red (R), green (G), and blue (B).
Currently, CNNs are the most successful Deep Learning architectures. CNN networks are composed of various types of layers. The most important is

the Convolutional layer. Other types of layers that are commonly used are pooling layers and non-linear layers. The output layer in CNNs is usually a fully-connected or dense layer, whose purpose is to combine the features extracted from the previous layers giving the final classification or regression output.
CNNs have achieved great success in processing multidimensional data, such as images, while RNNs have long been considered the reference architectures in the processing of one-dimensional temporal sequences of data.
Recently also CNNs have been widely applied to the study of sequential data.
S. Bai et al. propose an architecture called Temporal Convolutional Network (TCN),to combine simplicity, autoregressive prediction and very long memory [52]. TCN applies the principles of convolutional networks to the processing of sequential data, avoiding the classic problems of recurring networks, such as the vanishing / exploding gradient.
TCN is based on two fundamental principles, namely to create outputs of the same length as the input and to avoid future information from influencing the prediction of past information.
To satisfy the first principle, TCN consists of a 1D fully-convolutional network (FCN) architecture [53], where each layer of the network has the same input and output size.
To achieve the second principle, the TCN uses the so-called casual convolution, i.e. convolution occurs only to the left of the current instant. At time t only data from previous instants to t are taken into account.
This type of convolution is able to look backwards, to k data, where k is the filter size of the convolutional layer. This would not allow the network to work on time sequences, especially if they are long.
To overcome this problem, TCN employ dilated convolutions which allows the reception field to be expanded exponentially, while keeping a low complexity. Mathematically, for a one-dimensional input sequence x, and a filter dimension f, the dilated convolutional operation F on the elements x of the sequence k is defined as:

$$F(x) = (x * df)(s) \sum_{i=0}^{k-1} f(i) * x_{s-d*i} \tag{4.32}$$

where d is the dilation factor. For $d = 1$ the dilated convolution is equal to the standard convolution while for larger dilation factor the receptive field of the upper layers of the network widens; k is the filter size; $sd * i$ accounts for the direction of the past [52]. The TCN model used in this thesis is composed as shown in the Table 4.4.

Layer (type)	Param
tcn (TCN)	141376
dropout (Dropout)	0
dense (Dense)	65

Table 4.4: TCN model
Total params: 141,441
Trainable params: 141,441

The model is based on the use of the TCN implementation proposed in [54]. The most important parameters of this model are:
nb_filters: The number of filters to use in the convolutional layers. In this model each layer uses 64 filters;
kernel_size: The size of the kernel to use in each convolutional layer, a size of 3 was used in this work.
dilations: In this model it was used dilations=(1, 2, 4, 8, 16, 32). Each layer applies Rectified Linear Unit (ReLU) as a activation function. The model was trained for **100 training epochs** using the Adam gradient-based optimizer, a learning rate of 10^{-3} and a mini-batch size of **128 samples**. We used the **Mean Squared Error (MSE)** as a loss function.

Chapter 5

Experimental Results

This section shows the results obtained by the algorithms presented in section 4.2 in carrying out the TLE using the labeled dataset based on the accelerations measured by the sensors of the SHM described in section 3.1. Each algorithm was trained and tested to predict the number of light and heavy vehicles detected in each time window. As described in section 4.1.2, 9 datasets have been created that aggregate the accelerations detected by the sensors in different time windows, precisely of 5s, 6s, 7s, 8s, 20s, 30s, 40s, 50s and 60s.
These datasets were used to train and test four DL algorithms (simpleRNN, LSTM, GRU and TCN), while the other algorithms described in section 4.2 were trained and tested using the features extracted from these datasets (described in section 4.1.3). Datasets were randomly split using 70% for training and 30% for testing. In the section 5.1 are reported the performance of each algorithm in terms of Mean Absolute Error (MAE), that calculates the error by averaging the absolute error obtained in each window of the test dataset by applying the following mathematical formula:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y_i}|}{n} \tag{5.1}$$

Where y_i is the true vehicle count on the i-th window, $\hat{y_i}$ is the predicted vehicle count on the i-th window and n is the total number of windows. Moreover, are also reported the percentage mean absolute error (MAE%), which is obtained, as reported in equation 5.2, normalizing the Mean Absolute Error by the average true traffic load over all windows ($\bar{y}$). This metric is used to compare the performance of models in different time windows.

$$MAE\% = \frac{MAE}{\bar{y}} * 100 \tag{5.2}$$

Lastly, we also compute the coefficient of determination (R^2) score:

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.3}$$

Where y_i is the true vehicle count on the i-th window, $\hat{y_i}$ is the predicted vehicle count on the i-th window and $\bar{y}$ is the the average true traffic load (y) over all windows.
The R^2 score, provides an a-dimensional measure of how observed results are replicated by the model, representing the fraction of the variance explained by the model. It varies between -1 and 1, where 1 is the value obtained by an ideal regressor. In section 5.2 the performances of the various algorithms, described in section 5.1, will be compared.

5.1 Algorithm Performance

This section shows the performances of the regressors used in TLE. For the algorithms trained using the features, the tables describing the performance in the prediction of light and heavy vehicles will be reported. While for the algorithms trained using the raw data, only the porformances concerning the prediction of the heavy vehicle count are reported, because for the light vehicle count no algorithm has reached convergence.
For these algorithms the tables will show the results obtained using the raw dataset without applying scaling, applying standard scaling and applying MinMax scaling.

5.1.1 Linear Regressor (LR) Preformance

Table 5.1 shows the performance of the Linear Regressor in the TLE task. For each time window, the performances in the counting of heavy vehicles and light vehicles are reported.
We can see that in general the algorithm perform better in predicting the count of heavy vehicles than light ones, this is due to the fact that the passage of a heavy vehicle is more evident in the data collected by the sensors.

Window	**Heavy Vehicle**			**Light Vehicle**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,2	91,8	0,63	0,62	117,4	0,02
6s	0,23	71,8	0,68	0,67	94,9	-0,01
7s	0,28	68,3	0,67	0,74	98,2	-0,06
8s	0,21	56,8	0,76	0,69	81,3	0,06
20s	0,24	23,2	0,92	0,8	38,8	0,52
30s	0,26	16,5	0,94	0,75	23,3	0,82
40s	0,21	10	0,97	0,84	19,4	0,84
50s	0,24	9,5	0,97	0,78	13,7	0,88
60s	0,22	7	0,98	0,72	11,2	0,92

Table 5.1: Performance of the Linear Regressor in forecasting the count of heavy vehicles and light vehicles in the various time windows.

From the results it is evident that with the increase of the time window the results improve, both for heavy and light vehicles. We can see that in the prediction of the vehicle count in the small time windows (5s, 6s, 7s, 8s) the $MAE\%$ is very high for both light and heavy vehicles.
The worst result was obtained in the prediction of the light vehicle count in 5s time windows where $MAE\% = 117\%$.
In the large time windows (20s, 30s, 40s, 50s, 60s) the performances are decidedly better both in the counting of heavy vehicles and in that of light vehicles.
The best results have been achieved in the counting of heavy vehicles in the windows of 40s, 50s and 60s where the $MAE\%$ is less than 10% and the R^2 score approaches 1 (the ideal value).

5.1.2 Decision Tree (DT) Performance

Table 5.2 shows the performance of the Decision Tree in the TLE task. For each time window, the performances in the counting of heavy vehicles and light vehicles are reported.
We can see that in general the algorithm perform better in predicting the count of heavy vehicles than light ones. This is due to the fact that the passage of a heavy vehicle is more evident in the data collected by the sensors. From the results it is evident that with the increase of the time window the results improve, both for heavy and light vehicles.

Window	**Heavy Vehicle**			**Light Vehicle**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,19	58,8	0,45	0,58	96,8	-0,32
6s	0,15	51,6	0,55	0,55	95,3	-0,3
7s	0,14	49,8	0,63	0,63	92,1	-0,37
8s	0,18	49,3	0,57	0,72	86,8	-0,42
20s	0,23	24,6	0,77	0,78	36,3	0,42
30s	0,23	15,2	0,9	0,89	28,7	0,55
40s	0,17	8,3	0,94	0,82	19,5	0,70
50s	0,25	9,4	0,93	0,80	15,5	0,74
60s	0,21	6,6	0,95	0,87	13,4	0,78

Table 5.2: Performance of the Decision Tree in forecasting the count of heavy vehicles and light vehicles in the various time windows.

Observing the reported performances for heavy vehicles we can see that the $MAE\%$ varies from 59% in the time window of 5 seconds to 7% in the 60s one. The results in which the error is less than 10% can be considered excellent, therefore those obtained in the time windows of 40, 50 and 60 seconds. In fact, in these windows an R^2 score higher than 0.9 was obtained, thus approaching the ideal regressor which would have R^2 score equal to 1. As far as light vehicles are concerned, as mentioned the results are decidedly more disappointing, in the windows of 5, 6, 7 and 8 seconds there is an error of more than 90% compared to the average of the real values, with a negative R^2 score. Acceptable results are obtained for time windows ranging from 40 to 60 seconds, in which the $MAE\%$ is less than 20% and the R^2 score is between 0.7 and 0.8.

5.1.3 K-Nearest Neighbors (KNN) Performance

Table 5.3 shows the performance of the K-Nearest Neighbors in the TLE task. For each time window, the performances in the counting of heavy vehicles and light vehicles are reported.
As in the previous models, the KNN also obtains better performance in the prediction of heavy vehicles than light ones and the performance improves with the increase in the time window taken into consideration. From the results we can see that the $MAE\%$ in the prediction of heavy vehicles is always less than 60%. This model has also obtained excellent results in the

Window	Heavy Vehicle			Light Vehicle		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,19	58,5	0,63	0,52	87,2	0,24
6s	0,16	54,8	0,68	0,54	94,3	0,16
7s	0,13	47	0,78	0,56	82,1	0,18
8s	0,17	47,9	0,73	0,54	65,8	0,36
20s	0,25	26,1	0,85	0,58	27	0,77
30s	0,27	17,7	0,91	0,67	21,5	0,81
40s	0,27	13	0,94	0,57	13,6	0,9
50s	0,25	11,9	0,94	0,59	11,5	0,91
60s	0,27	8,6	0,96	0,58	9	0,92

Table 5.3: Performance of the K-Nearest Neighbors in forecasting the count of heavy vehicles and light vehicles in the various time windows.

prediction of light vehicles, in fact we can see that in the time windows of 40s, 50s and 60s, windows in which all the algorithms tested have obtained their best results, the performance in the prediction of the counting of heavy vehicles and light vehicles are practically the same. In fact, both predictions have an $MAE\%$ lower than 15% and an R^2 score higher than 0.9.
This model is the only one to have obtained a $MAE\%$ lower than 10% in the prediction of light vehicles, precisely obtained in the time windows of 60s.

5.1.4 Support Vector Regressor (SVR)

Table 5.4 shows the performance of the Support Vector Regressor in the TLE task. For each time window, the performances in the counting of heavy vehicles and light vehicles are reported.
As in the previous models, the SVR also obtains better performance in the prediction of heavy vehicles than light ones and the performance improves with the increase in the time window taken into consideration. This model showed excellent performance in the prediction of heavy vehicles, where it reached an $MAE\%$ lower than 15% in 4 dimensions of time windows (30s, 40s, 50s, 60s) also reaching an $R^2 \geq 0,95$. The performances are also very good in the prediction of the light vehicle count, where it obtained an $MAE\% < 20\%$ for time windows $\geq$ 40s and for time windows of 60s it obtained a $MAE\% = 11.8\%$ which can be considered a very good result.

Window	Heavy Vehicle			Light Vehicle		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,17	53,6	0,78	0,5	83,4	0,18
6s	0,17	59,9	0,76	0,44	77,1	0,34
7s	0,16	56,4	0,83	0,48	70,6	0,36
8s	0,18	48,3	0,83	0,5	61,1	0,41
20s	0,22	23,1	0,91	0,68	31,9	0,67
30s	0,22	14,6	0,95	0,79	25,4	0,76
40s	0,24	11,4	0,96	0,76	18,1	0,83
50s	0,26	9,8	0,97	0,89	17,2	0,82
60s	0,24	7,6	0,97	0,76	11,8	0,87

Table 5.4: Performance of the Support Vector Regressor in forecasting the count of heavy vehicles and light vehicles in the various time windows.

5.1.5 Multy Layer Perceptron (MLP) Performance

Table 5.5 shows the performance of the Multy Layer Perceptron in the TLE task. For each time window, the performances in the counting of heavy vehicles and light vehicles are reported.

Window	Heavy Vehicle			Light Vehicle		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,15	48,7	0,77	0,51	85,5	0,21
6s	0,14	47,7	0,78	0,50	79,8	-0,33
7s	0,13	47,1	0,83	0,49	72,7	0,38
8s	0,2	52,8	0,77	0,54	65,5	0,36
20s	0,23	23,5	0,9	0,67	31,3	0,77
30s	0,23	15,4	0,96	0,75	24	0,82
40s	0,26	12,4	0,96	0,92	22	0,8
50s	0,29	11	0,96	0,93	18,1	0,82
60s	0,25	7,8	0,98	0,98	15,2	0,83

Table 5.5: Performance of the Multy Layer Perceptron in forecasting the count of heavy vehicles and light vehicles in the various time windows.

in the previous models, the MLP also obtains better performance in the prediction of heavy vehicles than light ones and the performance improves

with the increase in the time window taken into consideration.
We can see from the results that the performance in the prediction of heavy vehicles is excellent for time windows of size ≥ 30s with a $MAE\% \leq 15.5\%$ and $R^2 \geq 0.96$.
In the prediction of the light vehicle count the error is high for the small time windows (5s, 6s, 7s, 8s) where the MAE % varies from 85.5 % to 65.5 %. Even on the large time windows, MLP does not perform very well, never obtaining a $MAE\% \leq 15\%$.

5.1.6 Standard Recurrent Neural Network (SimpleRNN) Performance

Table 5.6 shows the performance of the SimpleRNN in the estimation of the counting of heavy vehicles. For each time window, the performances using the raw data (No Scal.), the scaled data (Standard Scal.) and the normalized data (MinMax Scal.) are reported.

Window	**No Scal.**			**Standard Scal.**			**MinMax Scal.**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,36	133,5	-0,03	0,31	113,5	0,17	0,29	101,5	0,3
6s	0,38	107,3	0,09	0,35	114,2	0,25	0,28	100,8	0,39
7s	0,5	133,97	-0,26	0,36	89,03	0,24	0,32	91,68	0,42
8s	0,57	127,3	-0,06	0,44	92,2	0,22	0,35	74,1	0,51
20s	0,9	75,38	-0,04	0,76	78,88	0,13	0,68	68,03	0,24
30s	1,12	69,3	-0,01	1,02	63,72	0,21	1,23	77,94	-0,04
40s	1,32	64,62	-0,01	1,34	63,43	-0,17	1,39	67,29	0,01
50s	1,55	60,38	-0,01	1,62	60,43	0,06	1,52	58,71	0,01
60s	1,69	54,72	-0,01	1,73	54,52	-0,14	1,71	53,81	-0,01

Table 5.6: Performance of the Standard Recurrent Neural Network using the 3 version of the raw dataset in forecasting the count of heavy vehicles in 9 different time windows.

We can see in general that the algorithm does not perform well. In the smaller time windows (5s, 6s, 7s, 8s) the worst performances are obtained using the raw dataset, where we observe an $MAE\%$ around 120%. In these time windows the model obtains slightly better results when it is trained with the normalized dataset, where for 8 seconds time windows an average

error of 74% is obtained which is however very high and an R^2 score of 0.51, which for this metric is the best result obtained by the model.
In the larger time windows (20s, 30s, 40s, 50s, 60s) the algorithm obtains similar results for all 3 datasets, with an $MAE\%$ ranging from 80% to 55%. Although the RNN architecture is suitable for dealing with time series such as those present in the raw dataset, the performances obtained in this study were largely insufficient. This is certainly due to the fact that the dataset used is too small (only 31 minutes) to train an high complexity model.

5.1.7 Long-Short Term Memory (LSTM) Performance

Table 5.7 shows the performance of the LSTM in the estimation of the counting of heavy vehicles. For each time window, the performances using the raw data (No Scal.), the scaled data (Standard Scal.) and the normalized data (MinMax Scal.) are reported.

Window	**No Scal.**			**Standard Scal.**			**MinMax Scal.**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,41	151,2	-0,01	0,19	87,6	0,37	0,42	148,7	0,04
6s	0,48	145,8	-0,01	0,2	64	0,58	0,45	138,1	0,11
7s	0,48	147,9	-0,01	0,3	72,7	0,38	0,5	148,9	0,01
8s	0,59	128,9	-0,01	0,32	70,2	0,53	0,57	152,1	-0,01
20s	0,91	88,2	-0,01	0,64	63,5	0,34	0,83	84,8	-0,01
30s	1,12	71,2	-0,01	0,86	53,2	0,33	1,13	78	0,01
40s	1,45	69,7	-0,01	1,09	47,5	0,39	1,28	60,9	-0,01
50s	1,56	62,3	-0,02	1,29	47,8	0,24	1,48	59,4	-0,01
60s	1,77	55,77	-0,01	1,58	50	0,05	1,71	52,77	-0,01

Table 5.7: Performance of the LSTM using the 3 version of the raw dataset in forecasting the count of heavy vehicles in 9 different time windows.

As we can see from the results, the LSTM model also did not achieve sufficient results.
Using the raw dataset without scaling and the normalized raw dataset the results are similar to those obtained with the SimpleRNN with a very high $MAE\%$ in the small time windows (between 150% and 130%) and in any case high in the longer time windows, where the $MAE\%$ ranges from 90% to 55%.

Using the dataset scaled through the standard scaler, the results are better but still not sufficient, in fact we have a $MAE\%$ always higher than 45%.

5.1.8 Gated Recurrent Unit (GRU) Performance

Table 5.8 shows the performance of the GRU in the estimation of the counting of heavy vehicles. For each time window, the performances using the raw data (No Scal.), the scaled data (Standard Scal.) and the normalized data (MinMax Scal.) are reported. The GRU architecture performed better than

Window	**No Scal.**			**Standard Scal.**			**MinMax Scal.**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,40	149,8	0,01	0,21	97,5	0,39	0,2	91	0,48
6s	0,47	145,12	-0,01	0,24	75,3	0,54	0,27	81,8	0,47
7s	0,49	150,4	-0,01	0,33	79,7	0,33	0,27	80,5	0,53
8s	0,6	130,2	-0,01	0,36	79,3	0,37	0,24	63,2	0,7
20s	0,91	87,7	-0,01	0,71	70,3	0,22	0,41	42	0,71
30s	1,12	71,1	-0,01	1,04	63,7	0,15	0,47	28,9	0,86
40s	1,31	62,6	0,13	1,2	51,9	0,32	0,57	27,3	0,81
50s	1,56	62,1	-0,01	1,41	51,7	0,01	0,56	22,6	0,86
60s	1,06	33,5	0,58	1,69	53,4	-0,04	0,63	19,5	0,86

Table 5.8: Performance of the GRU using the 3 version of the raw dataset in forecasting the count of heavy vehicles in 9 different time windows.

the other RNNs tested. We can see that using the dataset without scaling the errors remain very high in the small time windows, where we have a $MAE\%$ ranging from 150% to 130%. In the larger time windows, the results are in line with the other RNN models, except than for the time window of 60s where an error percentage of 33% was obtained, therefore much better than the RNN models seen previously.
Using the data scaled through the standard scaler, the results are in line with those obtained for the other RNN models, therefore also in this case largely insufficient. The best results were obtained using the normalized dataset, where in the time windows of 30s, 40s, 50s, 60s the $MAE\%$ is less than 30% and the R^2 score values are high (between 0.81 and 0.86). We can say that this is the only case of acceptable predictions obtained by training networks with raw data.

5.1.9 Temporal Convolutional Network (TCN) Performance

Table 5.9 shows the performance of the TCN in the estimation of the counting of heavy vehicles. For each time window, the performances using the raw data (No Scal.), the scaled data (Standard Scal.) and the normalized data (MinMax Scal.) are reported.

Window	**No Scal.**			**Standard Scal.**			**MinMax Scal.**		
	MAE	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**	**MAE**	**MAE%**	**R^2**
5s	0,37	154,2	0,04	0,54	187,9	-1	0,25	104,3	0,24
6s	0,35	97,8	0,21	0,6	179,2	-0,9	0,28	84,9	0,38
7s	0,43	103,8	0,04	0,51	126,6	-0,23	0,32	99,1	0,29
8s	0,56	138,2	0,02	0,63	124,3	-0,26	0,31	72,2	0,53
20s	0,94	82,9	0,01	0,88	79,1	-0,03	0,65	66,6	0,37
30s	1,13	70,4	-0,01	1,28	83	-0,06	0,86	58,5	0,31
40s	1,37	63,7	-0,01	1,45	64,2	-0,07	1,34	64,5	0,02
50s	1,55	60,7	0,01	1,82	74,1	-0,25	1,4	51,7	0,05
60s	1,77	55,1	0,01	1,94	62,3	-0,63	1,71	56,2	-0,1

Table 5.9: Performance of the TCN using the 3 version of the raw dataset in forecasting the count of heavy vehicles in 9 different time windows.

The TCN algorithm was the algorithm that achieved the worst performance among all those tested. In fact, we can see an R^2 score of -1 obtained in the time window of 5 seconds using the scaled dataset, as we said above -1 is the worst result that the R^2 score can represent.
As we have seen in section 4.2.7 this algorithm is the most complex among those tested, with 141,441 trainable params. On the other hand, the dataset used to train it has proved to be insufficient insufficient to make the network converge.

5.2 Algorithm Comparison

This section compares the performance of the regressors tested in the TLE task.
Only the performances of the regressors trained using the extracted features are compared as we have seen in the previous section the regressors trained

with the raw dataset, did not obtain acceptable performances. This is due to the fact that the dataset used to train them was too small to allow highly complex models to converge.

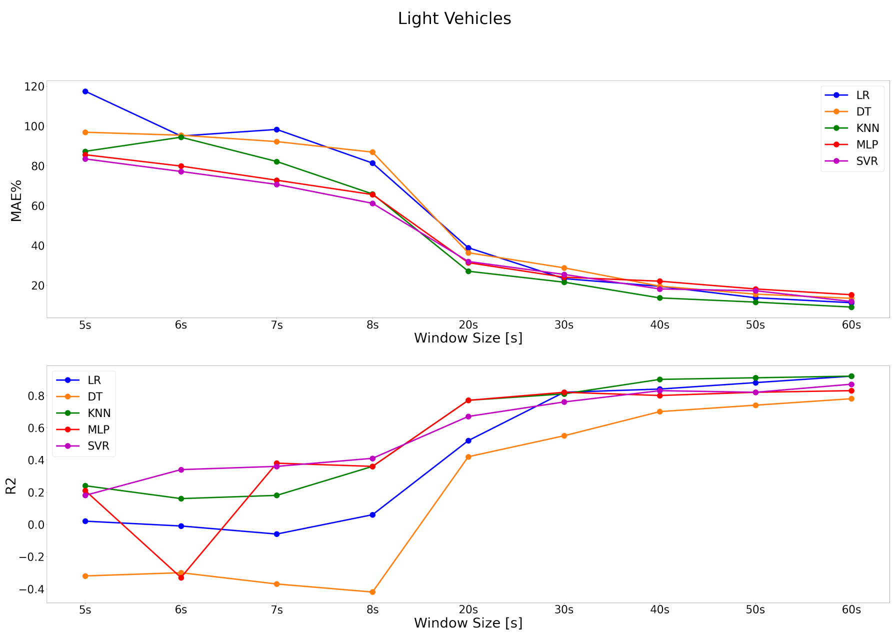


Figure 5.1: Window size versus performance for the tested regressor in the prediction of light vehicles count.

Figure 5.1 shows the performance of the regressors in predicting the count of light vehicles in terms of MAE% and R^2 score (R2 in the figure).
Each line represents the trend of the scores obtained by the specific regressors in the various Window sizes examined. As we mentioned in the previous section, it is evident that the performance of all the regressors improves with the increase in the size of the window on which the models are tested. We can see from the graph that for the smallest time window, 5s, the Linear Regressor (blue line) reports the highest MAE%.
In general we can see that in the smaller time windows (5s, 6s, 7s, 8s) the Linear Regressor and Decision Tree models have the worst performance both in terms of MAE% and in terms of R^2 score. In these windows the best results are obtained from the Multy Layer Perceptron and the SVR, which follow the same trend in terms of MAE% and have similar values in terms of R^2 score.
By examining the larger time windows (20s, 30s, 40s, 50s, 60s) we can see

how the results of all regressors improve and follow similar trends both in terms of MAE% and R^2 score. In these windows we can see that the KNN (green line) scores slightly better than the others both in terms of MAE% (it obtains 9% in the time window of 60s) and in terms of R^2 score, where it obtains R^2 scores higher than 0.9 for the windows of 40s, 50s, and 60s.
By examining the graph relating to the R^2 scores, we can see that the DT obtains the lowest scores in all time windows.

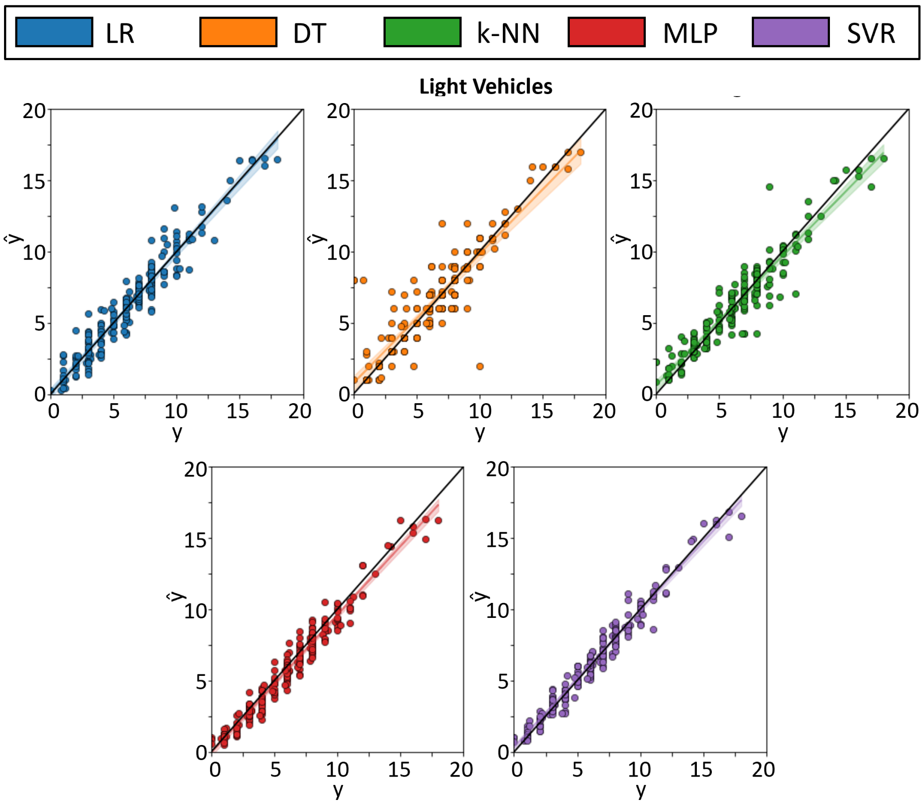


Figure 5.2: Real values versus predicted values on light vehicles. Colored areas represent 99% confidence intervals.

Figure 5.2 shows the comparison scatter plots between real counting values of light vehicles (y) and values predicted by the various algorithms ($\hat{y}$), in the time window of 60s, the window in which they obtained the best results. In an ideal situation, that is, where all the predicted targets are exactly identical to the real ones, all the points would be on the diagonal drawn in black. We can see that all the algorithms obtain very good performances having most of the points around the diagonal.

From the scatter plots we can see that the algorithms for which fewer outliers appear are the SVR and the MLP, while the DT is the one for which there are more points away from the diagonal as was predictable from the previously described metrics.

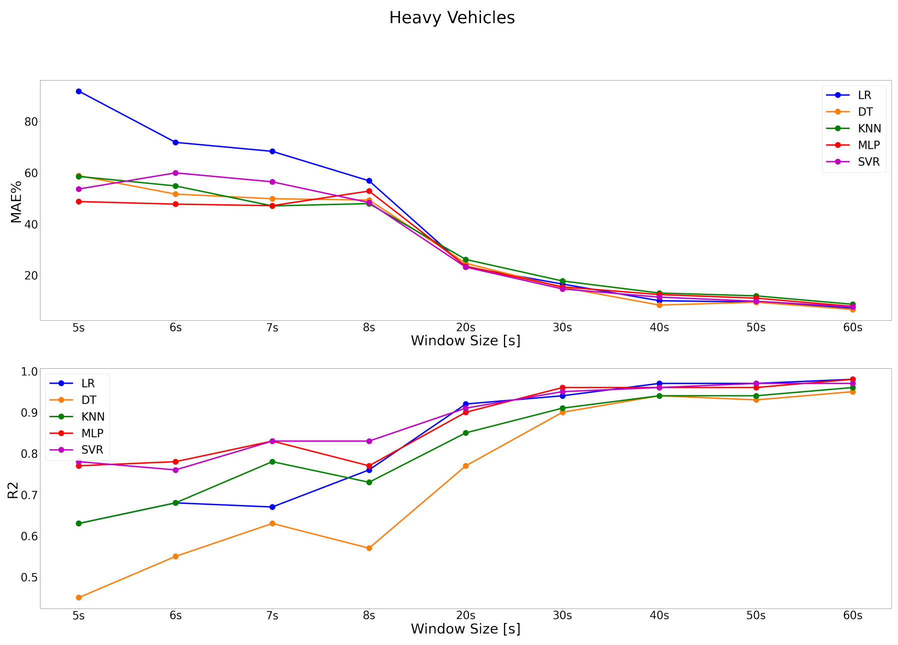


Figure 5.3: Window size versus performance for the tested regressor in the prediction of heavy vehicles count.

Figure 5.3 shows the performance of the regressors in the prediction of the heavy vehicle count in terms of MAE% and R^2 score (R2 in the figure). Each line represents the trend of the scores obtained by the specific regressors in the various Window sizes examined. Even in the case of heavy vehicle prediction, it is clear that the scores improve with the increase in the size of the time windows. In fact the MAE% decreases and the R^2 score increases. Observing the graph relating to the MAE% we can see how for the small time windows (5s, 6s, 7s, 8s) the Linear Regressor obtains the worst performance in terms of MAE%, which is clearly higher than the others. In the other hand, this result is not evident from the graph relating to the R^2 score. In fact the Decision Tree is the model that reports the lowest R^2 scores. This difference between the metrics, means that the Linear Regressor is the algorithm obtain the highest average of errors, while the Decision Tree makes more significant errors, predictions that present an high difference from the true value.

By examining the behavior of the other models in the small time windows, we can observe that the Multy Layer Percetron and the Support Vector Regressor obtain the best results in terms of R^2 score, while in terms of MAE% we can see that the SVR performs worse than the other models for time windows of 5s, and 6s.
Looking at the behavior of the models in the larger time windows (20s, 30s, 40s, 50s, 60s) we can see that there is a marked overall improvement in performance.
Also in this case, as happened in the smaller time windows, the Decision Tree has worse performance in terms of R^2 score than the other models. This distinction is clear in the 20s window, while for larger windows its behavior is very similar to that of the KNN.
We can see how the MAE% of all models in the large time windows follow a similar trend reaching very low error values, less than 15% for all models in the 40s, 50s, and 60s time windows.
Despite the lowest R^2 scores in all time windows, the DT is the model that achieves the best performance in terms of MAE% in the 60s window, where it reports an MAE% of 6.6%.
The best performance in terms of R^2 score was instead obtained by the MLP, which obtained an $R^2 = 0.98$, ranking first in this ranking.

Figure 5.4 shows the comparison scatter plots between real count values of heavy vehicles and values predicted by the various algorithms, in the time window of 60s, the window in which they obtained the best results.
We can see that all algorithms achieve very good performance, as demonstrated by the MAE% less than 15%. In the predictions of the DT we can see that there are some predicted values that differ considerably from the real values, especially for $y \leq 4$. This was predictable from the values of R^2, which despite being very good also for this model, are lower than to the scores obtained by the others.

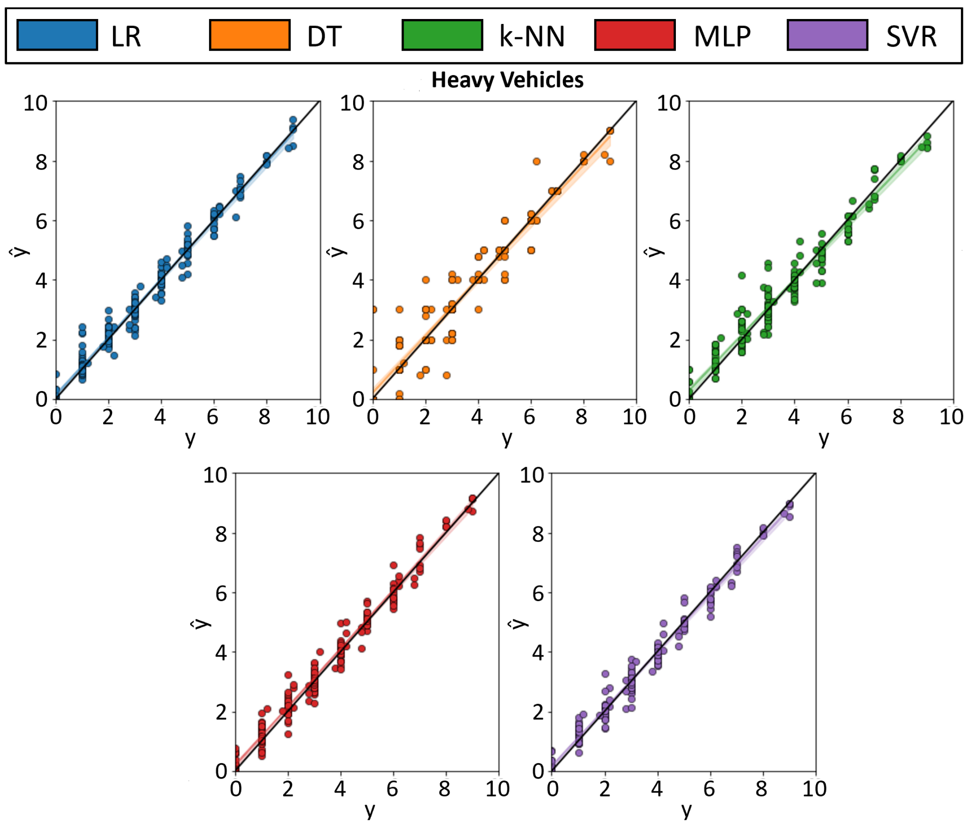


Figure 5.4: Real values versus predicted values on heavy vehicles. Colored areas represent 99% confidence intervals.

Chapter 6

Conclusions and Future Works

In this thesis, we presented the first application of supervised learning to traffic load estimation problem using data collected by accelerometers belonging to a SHM system. A pipeline has been proposed which includes a preprocessing phase, a supervised model training phase and finally a testing phase. Both classic ML models and DL models were trained and tested. The classic ML models, trained with the features extracted in the preprocessing phase, achieved the best performance. From the results, it is evident that all models perform better in predicting the heavy vehicle count than the light vehicle count. This is certainly due to the greater impact of heavy vehicles in the accelerations detected by the sensors. The results also show that the performance of the models improve with increasing the time window under consideration. In the time windows of size of 60 seconds all the classic ML regressors performed very well predicting the heavy vehicle count. Note that all models achieve a $MAE\% \leq 10\%$ and a $R^2 \geq 0.95$. Considering all the window sizes we can see that the SVR is the model that overall performs best in the prediction of heavy vehicles. In fact, the SVR is the only one to obtain an $MAE\% \leq 15\%$ in four window size (30s, 40s, 50s, 60s). In addition in all these four window sizes SVR reach a $R^2 \geq 0.95$. In the prediction of the light vehicle count, KNN was the model that obtained the best performance, obtaining a MAE% of 9% and an R^2 score of 0.92 for the windows with a size of 60 seconds. Deep Learning algorithms have not achieved convergence. This is due to the size of the dataset used, only 31 min, which has proven to be too small to allow highly complexity models to reach the right internal parameter setting. It would be interesting in future works to create a larger labelled dataset, in order to better train the proposed deep learning algorithms.

Bibliography

[1] Pasquale Cialdini, Corrado Loschiavo, Silverio Antoniazzi, and Vincenzo Torrieri. *Sistemi di monitoraggio del traffico linee guida per la progettazione.* Ministero delle infrastrutture e dei trasporti., pp. 15–16 (cit. on p. 1).

[2] S. Kamkar and R. Safabakhsh. «"Vehicle detection, counting and classification in various conditions"». In: *IET Intelligent Transport Systems, vol. 10, no. 6, pp. 406–413* (2016) (cit. on pp. 2, 16, 17).

[3] H. Dong, X. Wang, C. Zhang, R. He, L. Jia, and Y. Qin. «"Improved robust vehicle detection and identification based on single magnetic sensor"». In: *Ieee Access, vol. 6, pp. 5247–5255* (2018) (cit. on pp. 2, 16, 18).

[4] E. Odat, J. S. Shamma, and C. Claudel. «"Vehicle classification and speed estimation using combined passive infrared/ultrasonic sensors"». In: *IEEE transactions on intelligent transportation systems 19 (5) 1593–1606* (2017) (cit. on pp. 2, 16).

[5] Prashanth Mohan, Venkat Padmanabhan, and Ramachandran Ramjee. «Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones». In: *ACM Sensys.* Raleigh, NC, USA. Association for Computing Machinery, Inc., Nov. 2008. URL: `https://www.microsoft.com/en-us/research/publication/nericell-rich-monitoring-of-road-and-traffic-conditions-using-mobile-smartphones/` (cit. on p. 2).

[6] Ravi Bhoraskar, Nagamanoj Vankadhara, B. Raman, and Purushottam Kulkarni. «Wolverine: Traffic and road condition estimation using smartphone sensors». In: *2012 Fourth International Conference on Communication Systems and Networks (Comsnets 2012)* (2012), pp. 1–6 (cit. on p. 2).

[7] A. Burrello, D. Brunelli, M. Malavisi, and L. Benini. «Enhancing structural health monitoring with vehicle identification and tracking». In: *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)* (2020), pp. 1–6 (cit. on pp. 2, 15).

[8] Prof. Tom M. Mitchell. *For pioneering contributions and leadership in the methods and applications of machine learning.* Retrieved October 2, 2011. (Cit. on p. 5).

[9] Tom Mitchell. *Machine Learning.* McGraw Hill, 1997 (cit. on p. 5).

[10] J. Hu, Niu H., Carrasco J., B. Lennox, and Arvin F. «"Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning"». In: *IEEE Transactions on Vehicular Technology* (2020) (cit. on p. 5).

[11] Sebastian Raschka. *Introduction to Machine Learning and Deep Learning.* URL: `https://sebastianraschka.com/blog/2020/intro-to-dl-ch01.html`. (Aug 5, 2020) (cit. on p. 5).

[12] Aidan Wilson. *A Brief Introduction to Supervised Learning.* URL: `https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590`. (Sep 29, 2019) (cit. on pp. 7, 8).

[13] Amit Shekhar. *Understanding The Recurrent Neural Network.* URL: `https://medium.com/mindorks/understanding-the-recurrent-neural-network-44d593f112a2`. (Apr 14, 2018) (cit. on p. 10).

[14] John McGonagle, Christopher Williams, and Jimin Khim. *Recurrent Neural Network.* URL: `https://brilliant.org/wiki/recurrent-neural-network/` (cit. on p. 11).

[15] Alessio Burrello, Giacomo Sintoni, Davide Brunelli, and Luca Benini. «Zero-Cost Hardware Vehicle Traffic Estimation in Structural Health Monitoring». In: *Journal of latex class files, vol. 14, no.8* (2015) (cit. on pp. 13, 15, 21, 24).

[16] Giacomo Sintoni. «Monitoraggio del traffico attraverso segnali accelerometrici con l'utilizzo di algoritmi di Rilevamento delle anomalie». 2019 (cit. on pp. 13, 21, 24).

[17] Alessio Burrello, Alex Marchioni, Davide Brunelli, Simone Benatti, Mauro Mangia, and Luca Benini. «Embedded Streaming Principal Components Analysis for Network Load Reduction in Structural Health Monitoring». In: *IEEE Internet of Things journal, vol. X, no. X, X X* (2019) (cit. on p. 15).

[18] Y. Guo, Q. Zhang, K. K. Lai, Y. Zhang, S. Wang, and W. Zhang. «The impact of urban transportation infrastructure on air quality». In: *Sustainability 12* 14 (). DOI: 10.3390/su12145626 (cit. on p. 15).

[19] T. Afrin and N. Yodo. «A survey of road traffic congestion measures towards a sustainable and resilient transportation system». In: *Sustainability 12* 11 (). DOI: 10.3390/su12114660 (cit. on p. 15).

[20] T. Chen and C. Guestrin. «"Xgboost: A scalable tree boosting system"». In: *Proc. 22nd ACMSIGKDD Int. Conf. Knowl. Discovery Data Mining, pp. 785 794* (2016) (cit. on pp. 16, 19, 20).

[21] Q. Wang, J. Zheng, H. Xu, B. Xu, and R. Chen. «"Roadside magnetic sensor system for vehicle detection in urban environments"». In: *IEEE Transactions on Intelligent Transportation Systems 19 (5) 1365–1374* (2017) (cit. on p. 16).

[22] H. Liu, J. Ma, T. Xu, W. Yan, and X. Zhang L. Ma. «"Vehicle detection and classification using distributed fiber optic acoustic sensing"». In: *IEEE Transactions on Vehicular Technology 69 (2) 1363–1374* (2019) (cit. on p. 16).

[23] Z. Ye, H. Xiong, and L. Wang. «"Collecting comprehensive traffic information using pavement vibration monitoring data"». In: *Computer-Aided Civil and Infrastructure Engineering 35 (2) 134–149* (2020) (cit. on p. 16).

[24] Wu Y.N. Si Z. Gong H. et al. «"Learning active basis model for object detection and recognition"». In: *Int. J. Comput. Vis., 2010, 90, (2), pp. 198–235 (doi: 10.1007/s11263-009-0287-0)* (2010) (cit. on p. 17).

[25] T. Cerquitelli, D. Jahier Pagliari, A. Calimera, L. Bottaccioli, E. Patti, A. Acquaviva, and M. Poncino. «Manufacturing as a data-driven practice: Methodologies, technologies, and tools». In: *Proceedings of the IEEE 109 (4)* (2021), pp. 399–422. DOI: 10.1109/JPROC.2021.3056006 (cit. on p. 23).

[26] D. Cannizzaro, M. Zafiri, D. Jahier Pagliari, E. Patti, E. Macii, M. Poncino, and A. Acquaviva. «A Comparison Analysis of BLE-Based Algorithms for Localization in Industrial Environments». In: *Electronics 9 (1)* (2019), p. 44. DOI: `10.3390/electronics9010044` (cit. on p. 23).

[27] Travis E. and developers of NumPy. *Numpy Documentation.* URL: `https://numpy.org/doc/stable/` (cit. on p. 27).

[28] SciPy community. *SciPy Guide.* URL: `https://docs.scipy.org/doc/scipy/reference/stats.html` (cit. on p. 27).

[29] Bland JM and Altman DG. «Measurement error». In: *BMJ (Clinical research ed.) vol. 312,7047 (1996): 1654* (1996), p. 1654. DOI: `10.1136/bmj.312.7047.1654` (cit. on p. 27).

[30] Wikipedia. *Median.* URL: `https://en.wikipedia.org/wiki/Median` (cit. on p. 28).

[31] Amitava Dasgupta and Amer Wahed. «Chapter 4 - Laboratory Statistics and Quality Control». In: *Clinical Chemistry, Immunology and Laboratory Quality Control.* Ed. by Amitava Dasgupta and Amer Wahed. San Diego: Elsevier, 2014, pp. 47–66. ISBN: 978-0-12-407821-5. DOI: `https://doi.org/10.1016/B978-0-12-407821-5.00004-8`. URL: `https://www.sciencedirect.com/science/article/pii/B9780124078215000048` (cit. on p. 29).

[32] N. Unnikrishnan Nair, P.G. Sankaran, and N. Balakrishnan. «Chapter 3 - Discrete Lifetime Models». In: *Reliability Modelling and Analysis in Discrete Time.* Ed. by N. Unnikrishnan Nair, P.G. Sankaran, and N. Balakrishnan. Boston: Academic Press, 2018, pp. 107–173. ISBN: 978-0-12-801913-9. DOI: `https://doi.org/10.1016/B978-0-12-801913-9.00003-8`. URL: `https://www.sciencedirect.com/science/article/pii/B9780128019139000038` (cit. on p. 29).

[33] Luiz Paulo Fávero and Patrícia Belfiore. «Chapter 3 - Univariate Descriptive Statistics». In: *Data Science for Business and Decision Making.* Ed. by Luiz Paulo Fávero and Patrícia Belfiore. Academic Press, 2019, pp. 21–91. ISBN: 978-0-12-811216-8. DOI: `https://doi.org/10.1016/B978-0-12-811216-8.00003-3`. URL: `https://www.sciencedirect.com/science/article/pii/B9780128112168000033` (cit. on p. 29).

[34] NIST/Sematech. *NIST/Sematech e-Handbook of Statistical Methods.* NIST, https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm. DOI: `https://doi.org/10.18434/M32189` (cit. on p. 29).

[35] Thompson and Sylvanus P. *Calculus Made Easy.* Macmillan International, 1965, p. 185. ISBN: 9781349004874. DOI: `https://doi.org/10.18434/M32189` (cit. on p. 30).

[36] Jones and Alan R. *Probability, Statistics and Other Frightening Stuff.* Routledge, 2018, p. 48. ISBN: 781351661386. DOI: `https://doi.org/10.18434/M32189` (cit. on p. 30).

[37] Wikipedia. *Robust measures of scale.* URL: `https://en.wikipedia.org/wiki/Robust_measures_of_scale` (cit. on p. 30).

[38] scikit-learn developers. *sklearn API.* URL: `https://scikit-learn.org/stable/modules/classes.html` (cit. on pp. 31, 33, 34, 39).

[39] Jonathan Bartlett. *Deviance goodness of fit test for Poisson regression.* 2014. URL: `https://thestatsgeek.com/2014/04/26/deviance-goodness-of-fit-test-for-poisson-regression/` (cit. on p. 33).

[40] C. M. Bishop. *Pattern recognition and machine learning.* springer, 2006 (cit. on p. 34).

[41] Awad M. and Khanna R. *Support Vector Machines for Classification. In: Efficient Learning Machines.* Apress, Berkeley, CA, 2015. DOI: `https://doi.org/10.1007/978-1-4302-5990-9_3` (cit. on p. 37).

[42] Awad M. and Khanna R. *Support Vector Regression. In: Efficient Learning Machines.* Apress, Berkeley, CA, 2015. DOI: `https://doi.org/10.1007/978-1-4302-5990-9_4` (cit. on p. 38).

[43] Frank Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain.» In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 39).

[44] S. Abirami and P. Chitra. «Chapter Fourteen - Energy-efficient edge based real-time healthcare support system». In: *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases.* Ed. by Pethuru Raj and Preetha Evangeline. Vol. 117. Advances in Computers 1. Elsevier, 2020, pp. 339–368. DOI: `https://doi.org/10.1016/bs.adcom.2019.09.007`. URL: `https://www.sciencedirect.com/science/article/pii/S0065245819300506` (cit. on p. 39).

[45] Keras developers. *Keras API.* URL: `https://keras.io/api/` (cit. on pp. 41–44).

[46] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory.» In: *Neural Comput 9 (8)* (1997), pp. 1735–1780. DOI: `https://doi.org/10.1162/neco.1997.9.8.1735` (cit. on p. 42).

[47] Yang Xing, Chen Lv, and Dongpu Cao. «Chapter 8 - Driver Lane-Change Intention Inference». In: *Advanced Driver Intention Inference*. Ed. by Yang Xing, Chen Lv, and Dongpu Cao. Elsevier, 2020, pp. 193–233. ISBN: 978-0-12-819113-2. DOI: `https://doi.org/10.1016/B978-0-12-819113-2.00008-7`. URL: `https://www.sciencedirect.com/science/article/pii/B9780128191132000087` (cit. on p. 43).

[48] Shipra Saxena. *Introduction to Long Short Term Memory (LSTM)*. 2021. URL: `https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/` (cit. on p. 43).

[49] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: `1406.1078 [cs.CL]` (cit. on p. 43).

[50] Shipra Saxena. *Introduction to Gated Recurrent Unit (GRU)*. 2021. URL: `https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/` (cit. on p. 43).

[51] Simeon Kostadinov. *Understanding GRU Networks*. 2017. URL: `https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be` (cit. on p. 44).

[52] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. «An empirical evaluation of generic convolutional and recurrent networks for sequence modeling». In: *arXiv preprint arXiv:1803.01271* (2018) (cit. on pp. 45, 46).

[53] Jonathan Long, Evan Shelhamer, and Trevor Darrell. «Fully convolutional networks for semantic segmentation». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440. DOI: `10.1109/CVPR.2015.7298965` (cit. on p. 45).

[54] Keras developers. *Keras TCN*. URL: `https://github.com/philipperemy/keras-tcn` (cit. on p. 46).