CHALMERS
UNIVERSITY OF TECHNOLOGY

Discovering Patterns in Driving Data

Labeling and Classification of Common Driving Scenarios

Master's thesis in Engineering Mathematics and Computational Science

KAREN IVETTE BACA MENDOZA
WILHELM SÖDERKVIST VERMELIN

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Discovering Patterns in Driving Data

Labeling and Classification of Common Driving Scenarios

Karen Ivette Baca Mendoza
Wilhelm Söderkvist Vermelin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2019

Discovering Patterns in Driving Data
Labeling and Classification of Common Driving Scenarios
Karen Ivette Baca Mendoza
Wilhelm Söderkvist Vermelin

Supervisors: Chih Feng Lee, Propulsive Software Development, Volvo Cars AB
David Bolin, Department of Applied Mathematics and Statistics
Examiner: David Bolin, Department of Applied Mathematics and Statistics

Department of Mathematical Sciences
Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Discovering Patterns in Driving Data
Labeling and Classification of Common Driving Scenarios
Karen Ivette Baca Mendoza, Wilhelm Söderkvist Vermelin
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Modern vehicles are equipped with numerous sensors for providing feedback to the control unit. These measurements hold a substantial amount of information about the driver's action, environmental and traffic conditions.

In this thesis, we investigate using various machine leaning techniques to analyze driving data for discovering repetitive patterns when facing similar traffic situations. To this end, we first use unsupervised learning and data mining techniques to find driving patterns and to develop a labeling scheme. This last point consist of finding patterns in individual signals which are then combined to find patterns describing more complex behaviors. The discovered patterns and labels are used in the second part of the thesis to develop a classifier for recognizing the current driving situation. The classifier is designed such that it can be implemented in an Electric Control Unit of a production vehicle.

After the analysis, we were able to discover intelligible driving scenarios and we focused on some of them to label our data. We used this labeling to train and compare four different neural network architectures commonly used in time series classification. The models are trained by simulating an online situation where data comes in a form of data stream.

The results show that online classification is feasible. Implementing the classifier in the vehicle software could be beneficial for aiding the control unit in deciding gear shifts, energy recuperation and propulsion system. This may lead to a more efficient vehicle and a better driving experience.

Keywords: pattern discovery, machine learning, clustering, data mining, classification, neural networks, time series data, traffic scenarios, vehicle control.

Acknowledgements

We are very happy to have been given the opportunity to work on our thesis in the Propulsion Software Development Department at Volvo Cars. Here we want to spend a few words on thanking some people that have been important for this project. First of all, we wish to express our utmost gratitude to our Volvo supervisor Chih Feng Lee who has been immensely supportive and helpful during the course of this project. Working with Lee these past six months have truly been a pleasure. We thank him deeply for letting us work on such an interesting project.

We also wish to thank our academic supervisor David Bolin at the Applied Mathematics and Statistics Department at Chalmers University of Technology. David's input and knowledge have been very helpful and we are glad that he agreed to be our supervisor.

I, Karen, wish to thank my family and specially my mother for been so supportive and encouraged me all this time. I will be forever indebted with you and my father for all the opportunities that you have given me for continuing my studies. I am also thankful to my colleagues and friends that I have made during my time in Sweden. Thank you for listening and supporting me during this journey.

I, Wilhelm, wish to thank my family and friends for being supportive and helpful when writing this thesis. Special thanks to my brother Albert for being great friend and my partner Isabella for being caring and encouraging. I also wish to thank my mother, my father and my grandmothers for always believing in me.

Karen Baca Mendoza and Wilhelm Söderkvist Vermelin, 2019.

Contents

List of Figures

List of Tables

1
Introduction

In this chapter we describe the problem formulation, provide some background to the motivation of our project and establish a connection to the corresponding research area. In addition, we motivate the purpose, goal and delimitation of the work to be performed. Finally, we present an outline of the thesis.

1.1 Background

This thesis is carried out at the Propulsion Software Development department at Volvo Cars AB. One of the main responsibilities of this group is developing software for controlling and managing the power distribution in vehicles. The focus of this thesis will be on, but not limited to, the hybrid electric vehicle (HEV) which uses an electric motor and an internal combustion engine for propulsion. The two systems are more favorably used in different traffic situations. The electrical propulsion system is better used when cruising at low speed in cities and suburban areas, but can also generate a substantial amount of torque at an instant which can be exploited for quick acceleration. Propulsion via the internal combustion engine is advantageous in situations where the vehicle is maintaining a relatively constant high speed such as driving on the highway. Hence, we conclude that the optimal utilization of the propulsion systems is dependent on the traffic scenario.

The vehicle control unit is a micro-processor in the vehicle which, among other tasks, supervises the energy and torque distribution based on driver inputs such as braking and accelerating. It would be valuable to implement a subroutine in the software which classifies the current driving situation based on measurements collected from the most recent driving. Such a subroutine would aid the vehicle in controlling the propulsion system, gear shifting and power generation which could potentially reduce fuel consumption and battery drain.

The purpose of this thesis is to develop a method for traffic scenario discovery used for labeling and subsequently an online classification of the driving scenario.

The development of the algorithms will be based on driving information collected as time series by sensors in the vehicle. The data contains driver inputs and environmental conditions and it is described in detail in the next section. To analyze the time series and develop the methods used for achieving the thesis goals, various machine learning and data manipulation techniques will be utilized. In particular, methods suited for time series data will be explored.

1.2 Data Description

Table 1.1 shows the two types of features we work with in this thesis. The first type is defined as driving maneuvers and they are obtained from the data collection system in the car and recorded as a set of time series with a sampling frequency of 100 Hz. As its name suggests, these features are dependent on the driving behavior. The second type is classified as environmental factors which, contrary to the previously mentioned feature, are not dependent of the driver conduct and can be available a few seconds ahead of their occurrence.

The data set is composed by 22 driving logs of a route from Volvo Torslanda to Kungälv and back. This route is composed of a section inside the city center, where the speed limit is up to 50 km/h and a highway section where the speed limit reaches 100 km/h. In Figure 1.1 the signals recorded by the sensors in the vehicle are shown and in Figure 1.2 the route is shown.

Feature type	Name
Driving maneuvers	Vehicle speed [km/h]
	Accelerator pedal position [%]
	Brake pedal pressed [1/0]
	Steering angle [rad]
	Driver power demand [kW]
Environmental factors	Speed limit [km/h]
	GPS data [m×m]

Table 1.1: Features obtained from the on board data collection system in the test vehicle.

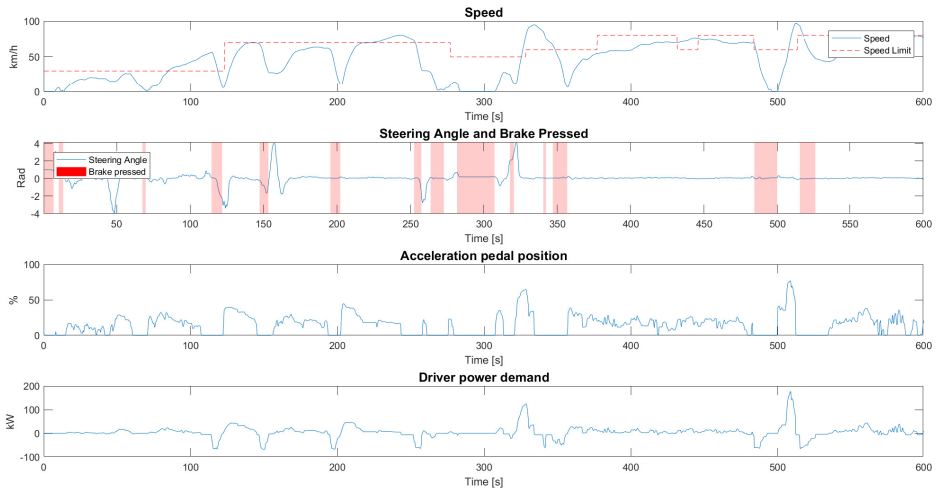


Figure 1.1: Example of the signals available for finding scenarios. The sample corresponds to 10 minutes of data gathering.

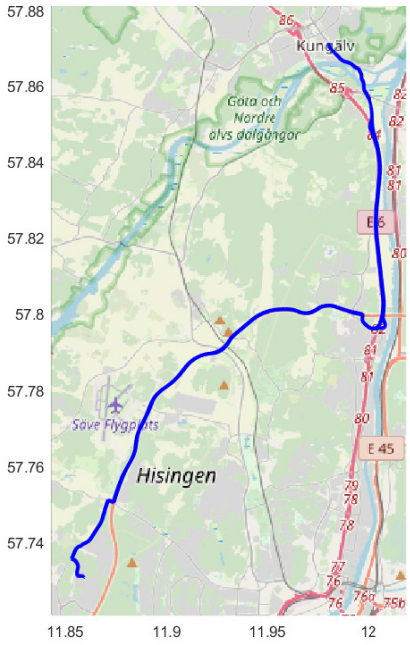


Figure 1.2: The route driven in the 22 driving logs.

1.3 Purpose

There are two main goals in this master thesis:

1. Discovering and labeling common driving scenarios.
2. Train a model for classifying the current driving situation.

The first goal will consist of two parts: first we discover patterns in individual signals which secondly, are combined to describe and discover more complex driver behaviors which will be the basis for labeling the data. The second goal is achieved by training a classifier on the labels obtained in the first part of the thesis. The classifier will be trained by simulating an online situation where information of the driving situation comes to the classifier as stream of data.

1.4 Delimitation

The analysis and the developed algorithms in this project are based on data collected when driving the route from Volvo Torslanda to Kungälv and back. No other drive cycle was analyzed which may have an effect on the performance of the model. The

data is only collected by one driver, which must be taken into consideration while assessing the robustness of the model.

1.5 Outline

This thesis contains five chapters. This first chapter we have formulated the problem, motivated the thesis objectives and described the project delimitation.

In the second chapter the theory necessary to achieve the thesis goals and obtain the desired results is introduced.

In the third chapter the methods and algorithms we implemented and developed to address the pattern discovery problem are described. This section also contains the results obtained from employing the methods on the driving data.

In the fourth chapter some of the patterns found are used for labeling the data. This labeling is then used for training four different neural network architectures. The training is performed in a manner that simulates online classification to facilitate the implementation in the vehicle software.

In the fifth chapter we discuss the results, lay out the implication of our work for future research and discuss the main points that can be learned from the research we have conducted.

2
Theory

In this chapter we introduce the theoretical and mathematical framework of the thesis. We explain the necessary statistical concepts for developing the methods used for attaining the thesis goals.

2.1 Time Series

The data collected by the control unit consists of the signals described in the previous chapter. The mathematical framework for analyzing such data is called *time series analysis*. Hence, we will dedicate this chapter to develop and describe common tools used for time series analysis and suitable machine learning techniques for clustering and mining temporal data.

2.1.1 Definition

A one-dimensional time series $\{T(t)\}_{t=t_1}^{t_N}$ is a sequence of N consecutive observations at time points $t_1, \ldots, t_N$. A multivariate time series is a collection of one-dimensional time series $\{[T_1(t), \ldots, T_p(t)]\}_{t=t_1}^{t_N}$ at time points $t_1, \ldots, t_N$. Again, N is referred to the number of observations and p is said to be the number of dimensions (or features) of the multivariate time series. The time points $t_1, \ldots, t_N$ do not necessarily need to be equidistant but since the data in this thesis are collected with a constant sampling frequency, we continue by assuming they are.

2.1.2 Change Point Segmentation

Segmentation refers to slicing the time series into several parts according to a specific criteria. One way to segment a time series is to cut it at points where there is a sudden change and is called *change point detection*. This particular algorithm finds an optimal segmentation of the time series, where the deviation of each segment from the desired estimate is minimized. The statistical estimate of interest could be for example, the mean, the standard deviation or the linear approximation of the segment. In steps, the algorithm finds a change point in the following way [1]:

1. partition the time series into two parts by splitting it at index k,

2. calculate an estimate for the desired statistic in each segment,

3. find the sum $J(k)$ of the point-wise deviations from the segment statistic estimate in terms of mean squared error (MSE),

4. vary the index k and redo steps 2 and 3 until $J(k)$ attains a minimum.

If we take the statistical measure to be the mean, the equation for $J(k)$ is given by

$$\begin{aligned} J(k) &= \sum_{i=1}^{k-1} \left(y_i - \frac{1}{k-1} \sum_{j=1}^{k-1} y_j \right)^2 + \sum_{i'=k}^{N} \left(y_{i'} - \frac{1}{N-k+1} \sum_{j'=k}^{N} y_{j'} \right)^2 = \\ &= (k-1)\mathrm{var}([y_1, \ldots, y_{k-1}]) + (N-k+1)\mathrm{var}([y_k, \ldots, y_N]). \end{aligned} \tag{2.1}$$

where $\mathrm{var}([y_m, \ldots, y_n])$ means the biased estimate of the sample variance of the slice of the time series y from index m to index n. Extending the above algorithm to finding more change points is just a matter of creating more partitions and finding their corresponding optimal indices. When generalizing the above method to other statistics, first note that the expression for $J(k)$ in equation (2.1) is of the form

$$J(k) = \sum_{i=1}^{k-1} \Delta(y_i; \chi([y_1, \ldots, y_{k-1}])) + \sum_{i'=k}^{N} \Delta(y_{i'}; \chi([y_k, \ldots, y_N])) \tag{2.2}$$

where Δ is a measure of the deviation from the segment statistic estimate χ. One statistic that will be interesting in a later part is the linear fit of each segment. The formula for $J(k)$ analogous to equation (2.1), in the case of the linear approximation can be given in terms of the segment deviations Δ and yields

$$\begin{aligned} \sum_{i=m}^{n} \Delta(y_i; \chi([y_m, \ldots, y_n])) &= (n-m+1)\mathrm{var}([y_m, \ldots, y_n]) + \\ &+ \frac{\left(\sum_{i=m}^{n} \left(y_i - \sum_{i=m}^{n} y_i \right) \left(\frac{n+m}{2} \right) \right)^2}{(n-m+1)\mathrm{var}([m, \ldots, n])}. \end{aligned} \tag{2.3}$$

2.1.3 Filtering

The presence of noise in a signal can complicate the analysis of the time series, so it may be beneficial reducing that noise as much as possible. One method of removing noise is by filtering the signal. There are many types of filters but arguably the simplest is the moving average filter.

This method uses a sliding window to smooth each signal value with the average of the neighboring values within the window size. For a window size of n_W the filtered signal $\tilde{y}$ from the original signal y is given by

$$\tilde{y}(k) = \frac{1}{n_W}(y(k) + y(k-1) + \cdots + y(k-n_W+2) + y(k-n_W+1)). \tag{2.4}$$

There are numerous ways of filtering data and we will not cover all here. For our intents and purposes the moving average filtering method is sufficient.

2.2 Data Representation

In order to facilitate the search of hidden patterns in the data, some methods for data transformation and dimensionality reduction will be revised in this section. Even though SAX is not implemented for the final results of the thesis, the technique is significant for the development of the pattern discovery algorithm in a later section.

2.2.1 Piecewise Aggregate Approximation (PAA)

Despite its simpleness, the PAA method is one of the most common procedures to reduce a time series $T(t) = t_1, ..., t_n$ into a lower dimensional vector $\bar{C} = \bar{c}_1, ..., \bar{c}_w$. The main procedure is to divide the data into w equal sub-sequences and replace it with the mean value $\bar{c}_i$ in each section [2]. Then, $\bar{c}_i$ can be calculated as:

$$\bar{c}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} t_j. \tag{2.5}$$

2.2.2 Symbolic Aggregate Approximation (SAX)

SAX is a method for discretizing a real-valued time-series by transforming the data into a vector of symbols [3]. After normalizing the time series and applying PAA dimensionality reduction on the signal, the values $\bar{c}$ are then grouped into β_{a-1} breakpoints. Assuming that the data after normalization follows a Gaussian distribution, the position of the breakpoints is calculated such that the area under the signal between β_{i-1} and β_i, $i = 1, ..., a-1$ is equal to $1/a$, where a is equal to the number of symbols used for the data transformation. A symbol is then assigned for each interval $[\beta_{i-1}, \beta_i)$, meaning that each symbol has the same probability to appear. An example of SAX transformation can be observed in Figure 2.1, where a time series of length $N = 1500$ is transformed into a string of length $w = 150$ using $a = 4$ symbols. The SAX sequence of the signal is then "*bddcaaabbccccccd*".

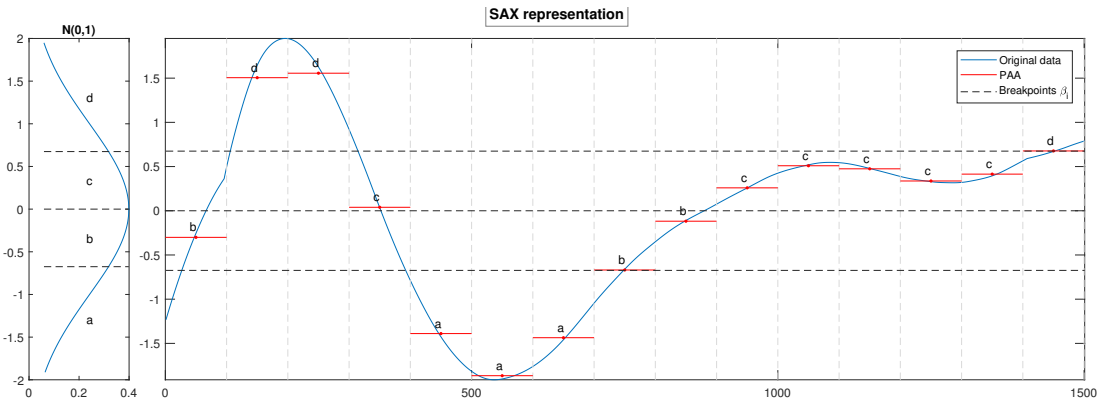


Figure 2.1: SAX representation of a time series of length $N = 1500$. the data is reduced to a vector of $w = 150$ and it is represented using 4 symbols.

2.3 Clustering

Clustering is the process of grouping similar observations. The similarity (or dissimilarity) of data points can be measured in various ways, e. g. their Euclidean distance, correlation or some other special measure that is suitable for the problem at hand. In this section we will describe the method for data clustering utilized for solving parts of the problems introduced in this thesis.

2.3.1 Similarity Measure

In order to compare the similarity between two objects, a distance measure must be defined. This is the main basis for distance-based clustering algorithms and it is also dependable of the type of analyzed data, for example, continuous or categorical. When measuring the similarity between two time series sub-sequences, we can find some issues such as difference in speed or length. *Dynamic Time Warping* is a suitable measure for this type of problem since it finds the alignment between the data points that minimizes the distance between the two signals.

Let $X = [x_1, \ldots, x_n]$ and $Y = [y_1, \ldots, y_m]$ define two sub-sequences of length n and m respectively. A matrix D of size $[n \times m]$ is constructed and each element contains the pairwise distance between the elements of the series $D(i, j) = d(x_i, y_j)$. In this case, $d(x_i, y_j)$ is the Euclidean distance between the points x_i and y_j [4]. Let $W = \{w_1, w_2, ..., w_K\}$, where $w_k = (i, j)$, represents the set of elements of the matrix defining the alignment of the elements x_i, y_j, and $\max(m, n) \leq K < m + n - 1$. Then, the DTW distance D_{DTW} can be described as an optimization problem:

$$D_{\text{DTW}} = \min_{W} \sum_{k=1}^{K} D(w_k) \tag{2.6}$$

Subject to the following constraints:

- Boundary conditions: $w_1 = (1, 1)$ and $w_K = (m, n)$
- Continuity: if $w_i = (a, b)$, then $w_{i-1} = (a', b')$, where $a - a' \leq 1$ and $b - b' \leq 1$
- Monotonicity: if $w_i = (a, b)$, then $w_{i-1} = (a', b')$, where $a - a' \geq 0$ and $b - b' \geq 0$

An example of the alignment of two time series can be observed in Figure 2.2. The plot in the center shows the optimal warping path obeying the constraints and achieving the alignment of the signals.

2.3.2 Types of Clustering

Using different criteria and initializations during clustering can lead to different results [5]. Clustering procedures are commonly divided in two types: *partitional* and *hierarchical* clustering. In the first type, the algorithm looks for an optimal partition

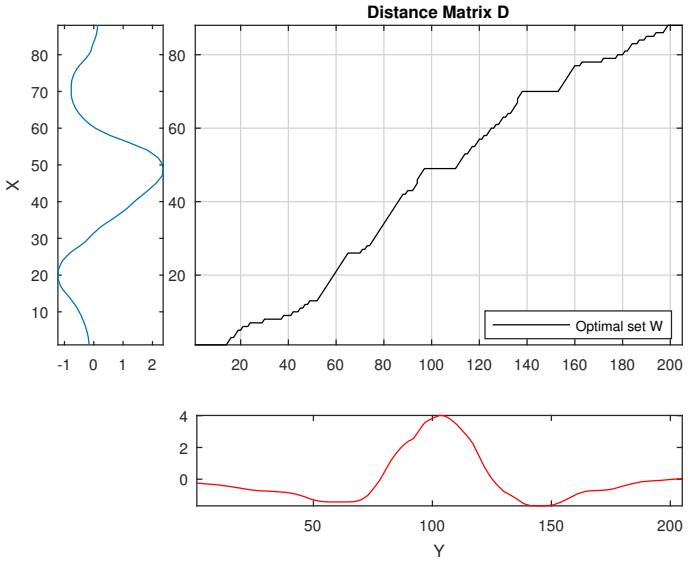


Figure 2.2: Alignment of the time series X and Y. The grid in the center represents the distance matrix between each of the data points. The black line is the optimal warping path that achieve the alignment of the signals by minimizing the distance.

of the data set into a specific number of groups by emphasizing certain characteristics such as density. In the second type, the algorithms prioritize clustering data according to their proximity. For hierarchical clustering, merging and splitting of clusters can be visualized using a *dendrogram*, which is a diagram in the shape of a tree. The node root represents the trivial clustering of putting the whole data set in the same cluster and the number of groups get increased as we go further down the tree. The last level of the dendrogram is called the *leafs* which is the other trivial extreme - each data observation is put in a separate cluster. By cutting the dendrogram at a specific level in the tree one decides the number of clusters for grouping the data. Hierarchical clustering algorithms are at the same time divided in two types. One of the them is known as *agglomerative* if the clustering is initialized with each observation in its own cluster and then the clusters are merged until satisfying some stopping criteria. Agglomerative methods are therefore also called *bottom-up* methods. The other type of hierarchical clustering algorithms is called *divisible methods* and here the algorithm initializes with all the data in the same cluster and progressively split clusters until terminating. Divisible methods are also called *top-down* methods.

In the thesis we have chosen to use Ward's minimum variance method [6] as the linking criteria when clustering the time series sub-sequences. This procedure is used for agglomerative clustering where the merging cost between two clusters, A and B, is defined as the increase of the within sum of squares given by

$$\begin{aligned}\Delta(A,B) &= \sum_{i\in A\cup B}(x_i - \bar{x}_{A\cup B})^2 - \sum_{i\in A}(x_i - \bar{x}_A) - \sum_{i\in B}(x_i - \bar{x}_B) \\ &= \frac{n_A n_B}{n_A + n_B}(\bar{x}_A - \bar{x}_B)^2,\end{aligned} \tag{2.7}$$

where n_j is the number of points in cluster j and $\bar{x}_j$ is the mean of the observations

in cluster j. The method stops merging clusters when it finds the minimum increase of the total within cluster sum of squares.

2.3.3 Evaluation

There are several methods for evaluating the clustering goodness-of-fit and analyzing the optimal number of clusters in a data set. A popular clustering evaluation method is the within sum of squares method [7] (also called "Elbow plot"), where the cluster within sum of squares (WSS) are plotted versus the number of clusters. The optimal number of clusters amounts to the number corresponding to the largest bend in the plot. The within sum of squares is defined in the following way for a data set of N observations and K clusters:

$$\text{WSS} = \sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik}(x_i - \bar{x}_k)^2, \quad \bar{x}_k = \frac{1}{n_k}\sum_{i=1}^{N} z_{ik}x_i, \tag{2.8}$$

where x_i is the ith data observation, $\bar{x}_k$ is the mean of cluster k, n_k is the number of observations in cluster k and z_{ik} is an indicator function, defined as

$$z_{ik} = \begin{cases} 1 & \text{if } x_i \in \text{cluster } k, \\ 0 & \text{if } x_i \notin \text{cluster } k. \end{cases} \tag{2.9}$$

There are clustering procedures where the WSS cannot be calculated as in Equation (2.8) due to the type data points used in the clustering, as for example, when clustering time series whose similarities are given by the DTW similarity matrix. In these occasions only the distance (or dissimilarity) matrix is available for calculating the WSS, which is done in the following fashion:

$$\text{WSS} = \sum_{k=1}^{K} \frac{(\mathbf{G}^\intercal \mathbf{D}\mathbf{G})_{kk}}{2n_k}. \tag{2.10}$$

In the above equation (2.10), $\mathbf{G}$ is a matrix whose elements are $(\mathbf{G})_{ij} = z_{ij}$, $\mathbf{D}$ is the element-wise squared distance matrix meaning, that if $\tilde{\mathbf{D}}$ is the original distance matrix, then $(\mathbf{D})_{ij} = (\tilde{\mathbf{D}})_{ij}^2$.

We see that from the definition in Equation (2.8), WSS is minimal for the case $K = N$ and maximal when $K = 1$. The largest bend in the graph is hard to define mathematically and usually requires visual inspection to determine. The rationale is that if there is a number K^* corresponding to a large bend in the graph, you gain a lot in terms of WSS up until K^* and gain notably less if the number of clusters is chosen higher than K^*. In Figure 2.3 we show an example of a WSS plot for an artificial data set where the known number of clusters is five.

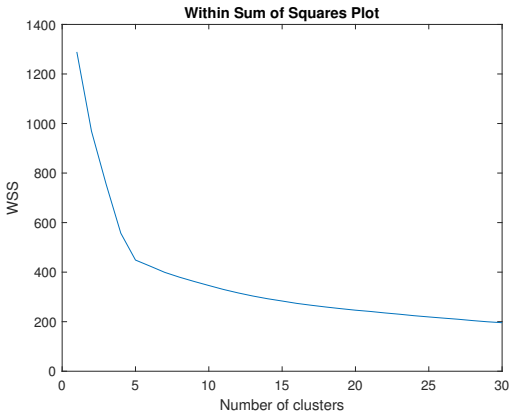


Figure 2.3: An example of a WSS plot for an artificial data set where the known optimal number of clusters is five, which corresponds well to the largest bend in the graph.

2.4 Pattern Discovery

In [8], Tanaka, Iwamoto and Uehara developed a method for the discovery of frequent patterns in multidimensional time series data. These sub-sequences of time series are called *motifs* and in this paper, they are able to recognize them even though they have different lengths. In order to work with multidimensional time series, they proposed to apply dimensionality reduction to obtain a one-dimensional time series. This reduction consists on the first principal component, since it is assumed that it keeps the most important information of the data. The dimensionality reduction is not implemented in the thesis, however, it is required a one-dimensional signal before the pattern discovery procedure.

After obtaining the one-dimensional data transformation, the number of points in the time series is reduced using PAA representation described in Section 2.2.1. This is performed by dividing the time series into sub-sequences of length $\frac{N}{w}$, where N is the original dimension of the time series and w the length after the reduction. The obtained vector $\bar{C} = \bar{c_1}, ..., \bar{c_w}$ is normalized and then transformed into SAX symbols. An example of a SAX string sequence can be observed in step i). This representation of the time series reduces the complexity of comparing continuous sequences of data by focusing mainly on the data trend. A window of size T_{min} is shifted around the SAX sequence and a *Behavior Symbol* (BS) is assigned to each unique combination. In step i a window size of $T_{min} = 3$ is defined and the 'aaa' SAX string is named Behavior Symbol A. The obtained BSs from the SAX sequence are listed in step ii. The collection of elements in the last column is called a *BS sequence* and it has a length of $w - T_{min} + 1$. The number of consecutive BSs are later stored in the BS length vector, as it can be observed step iii. The compressed sequence of BS symbols with no consecutive repetitions is called a *modified BS sequence* $\tilde{C}$. An analysis window of length w_{BS} is then shifted around $\tilde{C}$ with the purpose of finding BS patterns of length w_{BS}. Each of these are potential motifs of the time series. In order to choose the ones that better represent the data, a cost function based on the Minimum Description Length (MDL) principle is calculated. In general terms,

this rule is commonly used for model selection and statistical inference and it is based on the construction of a model that summarizes the data optimally regarding compression. Every BS pattern of length w_{BS} is then used to segment the modified BS sequence $\tilde{C}$, as it is done in step iv. The MDL cost function is calculated for each BS pattern and it is based on the cost of segmenting the sequence, its description length and the repetitions in the original time series. The pattern with the minimum cost function is then selected as the motif that better describes the data. We do not go into further detail about the MDL principle here since it is not implemented during the pattern discovery procedure in the thesis.

i) SAX sequence example, $T_{min} = 3$. The string 'aaa' defines the Behaviour Symbol A.

$$\overbrace{\underbrace{\mathrm{a\ a\ a}}_{\mathrm{A}}\ \mathrm{a}}^{\mathrm{A}}\ \mathrm{b\ c\ c\ a\ a\ b\ c\ c\ c\ d}$$

ii) BSs obtained from SAX sequence. The last column is called BS sequence and it has a length of $w - T_{min} + 1$.

SAX sequence			BS
a	a	a	A
a	a	a	A
a	a	b	B
a	b	c	C
b	c	c	D
c	c	a	E
c	a	a	F
a	a	b	B
a	b	c	C
b	c	c	D
c	c	c	G
c	c	c	G
c	c	d	H

iii) Modified BS sequence $\tilde{C}$. Consecutive repeated BSs are removed and the number of repetitions is saved in a vector.

BSs	A	B	C	D	E	F	B	C	D	G	H
Length	2	1	1	1	1	1	1	1	1	2	1

iv) Obtained segments around the '*BCD*' pattern. In this example $w_{BS} = 3$.

A	***B***	***C***	***D***	E	F	***B***	***C***	***D***	G	H
2	1	1	1	1	1	1	1	1	2	1

2.5 Classification Model

For the second part of this thesis a classifier model will be implemented in order to classify the current traffic scenario from the labels obtained in the first part. At this point the problem has been reduced to a time series classification problem, where each time point is associated to a label corresponding to a particular traffic scenario. In the next section we will discuss neural network classifiers and several popular neural network models for time series classification.

Classification is a part of supervised machine learning where each data point or observation $\mathbf{x}$ is associated to one unique output $\mathbf{y}$ meaning there is a surjective mapping

$$\mathbf{x} \mapsto \mathbf{y} \tag{2.11}$$

between each observation and each output. Outputs are sometimes called labels, classes or targets and the terms will be used interchangeably here. Classification is the task of estimating the mapping (2.11). The estimate of the output $\mathbf{y}$ is given by $\hat{\mathbf{y}} = f(\mathbf{x})$ where f is the classification model. In machine learning, the model is fitted to the data by training the model on the data set which means that the model is shown observations, maps them to an output estimate $\hat{\mathbf{y}}$ and the parameters of the classifier are updated according to some loss function, $\ell(\hat{\mathbf{y}}, \mathbf{y})$, of the estimated output $\hat{\mathbf{y}}$ and the actual output $\mathbf{y}$. The model is trained until ℓ is minimized. Usually when training a classifier some data is held out and thus not trained on, in order to validate the true performance of the classifier after training is done. This prevents the classifier from learning structures in the data that comes from random noise and thereby gives a more robust model.

2.5.1 Metrics

A metric is defined as a measure of performance and in our context, it is used for estimating the classification performance. These measurements can help to evaluate different characteristics of a model after the training is done. Another common use of metrics is for model selection. When comparing the performances of the trained models in this thesis, we use threshold types of discriminator metrics to evaluate the classification of the test data. An imbalance of the number of observations in each class can cause a poor performance when classifying members of the minority classes. This is the reason why we need to evaluate our models with a metric that takes this issue into account and better reflect the model performance than just accuracy. In the list below, some of the common metrics are defined. For a simpler interpretation, we make use of the confusion matrix, shown in Table 2.1. The rows in the table represent the actual class labels while the columns correspond to the classification results. True positive (TP) and true negative (TN) predictions in a binary classification are observations that were correctly positively and negatively classified, respectively, while false positive (FP) and false negative (FN) predictions represent the elements identified as positive and negative, respectively, when the real labels are actually the opposite [9]. To clarify, let us use a simple example of cancer detection where the label 0 represents that the patient is healthy and 1 represents

the opposite. In this example a true positive or true negative classification means correctly classifying that a person is sick or not sick, respectively. The false positive classification means in this case that we classify a healthy person as sick and the false negative classification means that the illness was not detected and the patient is classified as healthy. This example makes it clear that it is important to keep track of these measures, since a classifier can still perform quite well while still having a high false negative rate. For example, in the context of cancer detection a high false negative rate is completely unacceptable.

		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

Table 2.1: A binary classification confusion matrix.

Below four commonly used metrics are defined:

- **Accuracy**
  Measure based on the total number of instances correctly identified. Correct classes are treated equally, so this type of metric can be used to measure performance for data sets with a balanced number of classes. Numerically, it can be calculated as

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{2.12}$$

- **Precision**
  Probability of the patterns correctly predicted as positive when they were actually positive. A common example of requiring a high precision rate over recall is in the identification of spam mails. Important mail correctly identified as suspicious (false positive) can go to the junk folder and be erased. This metric can be calculated as

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{2.13}$$

- **Recall**
  Fraction of positive patterns correctly identified as positive. A common example of a high recall requirement is in the detection of deceases, since it is aimed for reducing the number of false negative results. False negative represents the number of sick people that are diagnosed as healthy, so there is a risk that they will receive no treatment. Recall is defined as follows

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{2.14}$$

- **F1-score**
  It is defined as the harmonic mean of precision and recall. It is used as a trade-off for measuring performance of classification where the imbalance problem is present. When using the F1-score false positive and false negative results are equally penalized. Mathematically, it can be described as

$$\text{F1-score} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}. \tag{2.15}$$

2.6 Artificial Neural Network

Artificial neural networks (ANNs) are inspired by the biological neural networks such as those in the nervous system in animals [10]. ANNs have proven to perform well in many areas of machine learning and in particular supervised learning. The canonical example of a neural network is the *Multi-Layer Perceptron* (MLP). The basic structure of an MLP is layers of neurons in sequence where each neuron in a layer is connected to each neuron in the next one. The connections in a neural network are referred to as *weights*. The MLP is a type of *feed-forward* neural network, which means that there are no cycles in the neural weight connections, i.e., a neuron only connects to neurons in the next layer and never to a neuron in a previous layer or itself. The first layer is called the *input layer* since the data observations $\mathbf{x}$ are fed through this layer and the last one is called the *output layer*, where each neuron represents an element in the estimate $\hat{\mathbf{y}}$. The layers in between the input and the output are called *hidden layers* and they are hidden in the sense that the input and output of those layers are never seen or interpreted in the training phase. These layers only exists to enable the non-linearity of the mapping $\hat{\mathbf{y}} = f(\mathbf{x})$. In fact, an MLP with one hidden layer with a finite number of neurons is a "universal approximator", meaning that it can approximate any continuous function on compact subsets of $\mathbb{R}^n$ under some weak assumptions on the activation functions of the neurons [11]. The *activation function* g is the non-linear transformation of data that passes through the neuron and it is usually a "squashing function" such as the sigmoid function $\sigma(x)$ or $\tanh(x)$ given by

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.16}$$

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \tag{2.17}$$

These activation functions are mostly used in shallow networks, i. e., networks with only a few hidden layers. For deep neural networks, there is another popular activation function which is called the Rectified Linear Unit (ReLU) activation function [12] given by

$$g(x) = \max(0, x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases} \tag{2.18}$$

The reason for the popularity of ReLU activation functions in deep neural networks is due to their tendency of reducing a common problem in the training phase called the

vanishing gradient problem [13] which we will not go into detail about. However, the vanishing gradient problem is a larger concern for deep neural networks and causes the network to stop learning which, of course, is undesirable. We will use ReLU activation functions in the neural network implementations in this thesis so it is still of interest to mention them here. The last activation function commonly used in classification is called the softmax function which is defined as

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j^K \exp(z_j)}, \quad \mathbf{z} \in \mathbb{R}^K. \tag{2.19}$$

Note that

$$\text{softmax}(\mathbf{z})_i \in (0,1), \quad \forall i \in \{1,\dots,K\}, \tag{2.20}$$

meaning that the output of the softmax function can be interpreted as a probability over K classes, which makes it useful for classification.

The mappings given by the connections in the MLP is calculated as a weighted sum of the inputs from the previous layer, which is why the connections are referred to as "weights". Let $h_j^{(i)}$ be the jth neuron in the ith layer in the network, where $j \in \{1,\dots,l_i\}$ (l_i being the number of neurons in layer i). Then its mapping is given by

$$h_j^{(i)} = g\left(\sum_k^{l_{(i-1)}} w_{jk}^{(i)} h_k^{(i-1)} - \theta_j^{(i)}\right), \quad j = 1,\dots,l_i, \tag{2.21}$$

where g is the activation function (usually sigmoid for MLPs), $w_{jk}^{(i)}$ is the weight connecting neuron k in layer $i-1$ with neuron j in layer i and $\theta_j^{(i)}$ is called the *bias* of neuron j in layer i. Naturally, $h_j^{(0)} = x_j$ and $h_j^{(L-1)} = \hat{y}_j$ for a network with L layers. For an output layer with softmax activation and L number of layers, $\hat{y}_j$ is given by the following equation

$$\hat{y}_j = \text{softmax}(\mathbf{b}^{(L-1)})_j = \frac{\exp\left(b_j^{(L-1)}\right)}{\sum_k^K \exp\left(b_k^{(L-1)}\right)}, \tag{2.22}$$

where $\mathbf{b}^{(L-1)}$ is a vector with components given by

$$b_j^{(L-1)} = \sum_k^{l_{(L-2)}} w_{jk}^{(L-1)} h_k^{(L-2)} - \theta_j^{(L-1)}, \quad j = 1,\dots,K. \tag{2.23}$$

Hence, $\hat{\mathbf{y}} = [\hat{y}_1,\dots,\hat{y}_K]$ is a collection of probabilities over the K classes. The final prediction for a particular class $\hat{c} \in \{1,\dots,K\}$ is then obtained by finding the index of the maximum probability in $\hat{\mathbf{y}}$:

$$\hat{c} = \arg\max \hat{\mathbf{y}}. \tag{2.24}$$

In Figure 2.4 a simple architecture of an MLP with two hidden layers is shown. This represents the mapping $\hat{\mathbf{y}} = f(\mathbf{x})$.

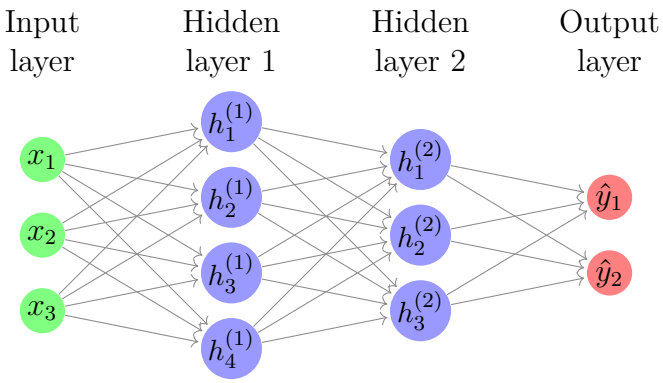


Figure 2.4: The architecture of an MLP with two hidden layers.

A feed-forward neural network is trained by updating the weights and biases in the network to minimize the loss of the output. Specifically, the weights and biases are updated by means of stochastic gradient descent on a loss-function utilizing the chain-rule. The loss implicitly depends on the weights and biases in the network due to the connections between neurons. The update rule (gradient descent) can be stated as

$$\tilde{\mathbb{w}} = \mathbb{w} - \eta \frac{\partial \ell(\hat{\mathbf{y}}, \mathbf{y}; \mathbb{w}, \Theta)}{\partial \mathbb{w}}, \tag{2.25}$$

$$\tilde{\Theta} = \Theta - \eta \frac{\partial \ell(\hat{\mathbf{y}}, \mathbf{y}; \mathbb{w}, \Theta)}{\partial \Theta}, \tag{2.26}$$

where $\mathbb{w}$ and Θ are the weights and biases in the network while $\tilde{\mathbb{w}}$ and $\tilde{\Theta}$ are their respective updates. Here η is a parameter called the *learning rate* which corresponds to the step length in the gradient descent algorithm and hence the rate at which the network "learns".

2.6.1 Loss function

As briefly mentioned before, a loss function (also known as a cost function) measures the error of the estimated targets $\hat{\mathbf{y}}$ in relation to the true targets $\mathbf{y}$. One simple loss function is the squared error loss function given by $\ell(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}(\mathbf{y} - \hat{\mathbf{y}})^2$. One realizes easily that $\ell(\hat{\mathbf{y}}, \mathbf{y})$ takes its minimum value $\ell = 0$ at $\hat{\mathbf{y}} = \mathbf{y}$, which means that as the loss function is minimized, the estimate $\hat{\mathbf{y}}$ becomes closer to $\mathbf{y}$. This property is desirable for any loss function since the estimated targets $\hat{\mathbf{y}}$ should get closer to the real classes $\mathbf{y}$ as the training progresses. The squared error loss function has the advantage of having nice mathematical properties such as smoothness and symmetry. One of the most widely used loss functions in classification is the categorical cross-entropy given by

$$\mathrm{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{K} \omega_i y_i \log(\hat{y}_i), \tag{2.27}$$

where K is the number of classes, $y_i \in \{0, 1\}$ is the one-hot encoding of the class labels given by

$$y_i = \begin{cases} 1 & \text{if input } \mathbf{x} \text{ has label } i, \\ 0 & \text{otherwise,} \end{cases} \tag{2.28}$$

and $\hat{y}_i$ is the estimated probability of the input $\mathbf{x}$ belonging to class i. To deal with an imbalance in the number of observations in each class, weights $\omega_i \in [0, 1]$ are introduced. These weights are inversely proportional to the frequency of the classes, meaning that misclassification of the least frequent class is penalized the most. The class imbalance problem arises when some classes dominate others in terms of their sample proportions. Then the classifier is more likely to assign the most frequent classes since they are the most probable guess. In the worst case, the classifier does not predict anything other than the most common class because it may still achieve a high accuracy thus rendering the model useless.

Another way to address the class imbalance problem is by implementing the focal loss function given by

$$\mathrm{FL}(\hat{\mathbf{y}}, \mathbf{y}) = -\alpha \sum_{i=1}^{K} y_i (1 - \hat{y}_i)^{\gamma} \log(\hat{y}_i), \tag{2.29}$$

where $\gamma \geq 0$ is called the focusing parameter and its recommended values are $\gamma \in [0, 5]$ according to [14]. The parameter α is introduced for numerical stability when implementing the function in software. The advantage of the focal loss is the effect achieved with the so called modulating factor given by $(1 - \hat{y}_i)^{\gamma}$. This coefficient helps to regulate the contribution of the samples that are easily correctly identified. A sample is "easy" to classify if the estimated probability $\hat{y}_i$ is close to one when the the sample belongs to class i. On the other hand, when the sample is misclassified and $\hat{y}_i$ is close to zero, the modulating factor is close to one and the loss function reduces to the categorical cross-entropy function. High expected probabilities have a low modulating factor, reducing the penalization.

Using some of the loss functions introduced in this section, situations may arise where expressions amount to $0 \log 0$ which, strictly mathematically speaking, is not defined. In these cases $0 \log 0$ is replaced with zero since

$$\lim_{x \to 0} x \log x = 0. \tag{2.30}$$

2.6.2 Recurrent Neural Network (RNN)

An RNN is a network with connections like in the MLP but also including adjacent edges, known as recurrent edges, representing time steps [15]. This property make them suitable for processing sequential data, enabling its use for the time series classification (TSC) task. At time t, the nodes with recurrent edges receive the current input $\mathbf{x}_t$ and also previous ones from hidden layers at previous times h_{t-1}, as it can be observed in Figure 2.5. This means that the output at one point can be affected by consecutive previous samples. The hidden state h_t at time t is given by

$$h_t = g\left(W_{ih} x_t + W_{hh} h_{t-1} - \theta_h\right), \tag{2.31}$$

where g represents the activation function, W_{ih} the conventional weights connecting the inputs and W_{hh} the recurrent connections between the hidden layers. As in the MLP, the function also includes a bias parameter and in this case is represented by θ_h. A general architecture of an RNN can be observed in Figure 2.6, where the important characteristic is the recurrent connection for the hidden state.

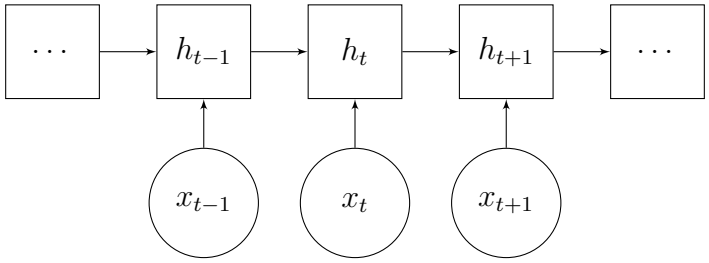


Figure 2.5: Unfolded representation of an RNN, showing the recurrent edge connections and its dependence of previous time steps.

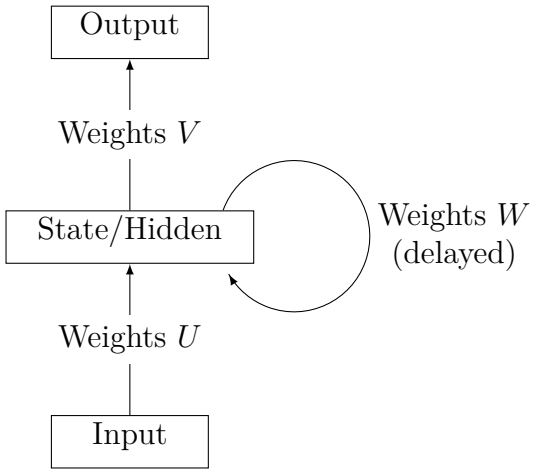


Figure 2.6: General architecture of an RNN. The hidden layer contains the information of sequential data.

2.6.3 Long Short-Term Memory (LSTM) network

LSTM networks are a type of gated RNNs introduced in 1997 [16]. LSTM networks have the benefit of being able to retain longer sequential patterns and remember its past hidden states. This happen in part due to having a memory cell connection resulting in a longer short term memory than a regular RNN (hence its name). The structure of a LSTM unit is similar to a RNN in the sense that previous predictions are used as input to the network, but the LSTM has some additional structure, in particular, the LSTM cell has three gates, the forget, input and output gates, and a memory cell. Arguably the most important gate is the forget gate which is controlling the connection of the memory cell to the output. The memory cell contains information about previous predictions and their hidden states. The input gate can be thought of as an ignoring gate or the attention of the unit, controlling

which inputs are important for the current prediction. The output gate acts as a filter where selection of which memories, previous predictions and new information should be prioritized for a particular prediction. The inputs to a gate are the previous predictions and new data after both have been transformed by a neural network which is simultaneously being trained. The action of the gate is that an element-wise vector multiplication (the Hadamard product) is performed between the different inputs to the gate. The architecture of an LSTM is shown in Figure 2.7. In equations, the different calculations performed in the LSTM unit are given by

$$\mathbf{i}_t = g_1\left(\mathbb{W}_i\mathbf{x}_t + \mathbb{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i\right), \tag{2.32a}$$
$$\mathbf{f}_t = g_1\left(\mathbb{W}_f\mathbf{x}_t + \mathbb{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f\right), \tag{2.32b}$$
$$\mathbf{o}_t = g_1\left(\mathbb{W}_o\mathbf{x}_t + \mathbb{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o\right), \tag{2.32c}$$
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ g_2\left(\mathbb{W}_g\mathbf{x}_g + \mathbb{U}_g\mathbf{h}_{t-1} + \mathbf{b}_g\right), \tag{2.32d}$$
$$\mathbf{h}_t = \mathbf{o}_t \circ g_2(\mathbf{c}_t), \tag{2.32e}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ are the activation vectors of the input, forget and output gates at time step t. The input and recurrent weight matrices are denoted as $\mathbb{W}_r$ and $\mathbb{U}_r$, respectively. Additionally, $\mathbf{b}_r$ denotes the bias vectors. Here the subscript r is either i, f, o or g representing the connections in the input, forget, output or memory cell gates, respectively. The activation function is denoted by g_1 and g_2 where g_1 is the activation function for the recurrent steps and g_2 is the activation function used for updating the cell state or the memory state. The element-wise multiplication is denoted by $\circ$. Lastly, $\mathbf{h}_t$ represents the internal state of the LSTM unit and $\mathbf{c_t}$ is the memory cell state at time step t. Usually one initializes the states of the LSTM unit at $t = 0$ as $\mathbf{c}_0 = \mathbf{h}_0 = \mathbf{0}$.

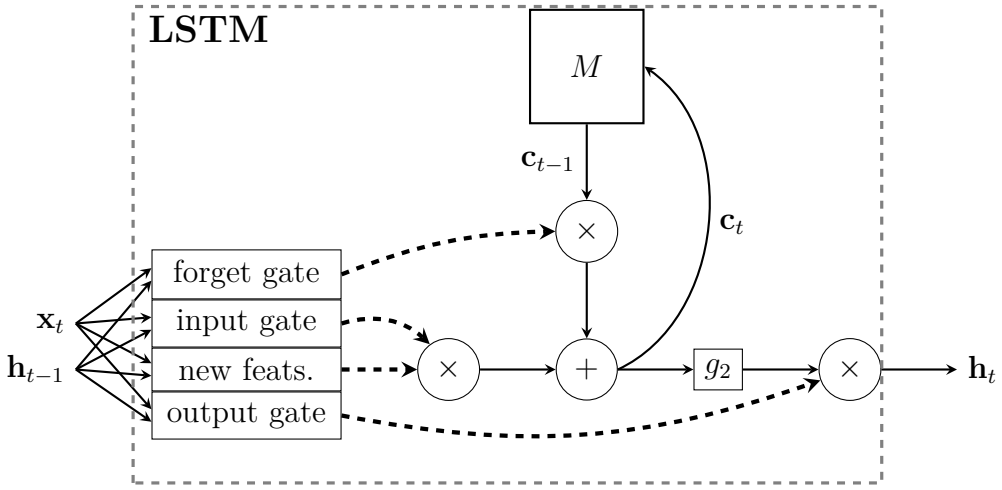


Figure 2.7: The architecture of an LSTM cell. The nodes marked with $\times$ represent a gate with element-wise multiplication of the inputs and + represent element-wise addition of the inputs. The box containing g_2 represents applying some activation function g_2 to the incoming data and $\mathbf{c}_t$ represent the internal state of the memory cell M for input time t. The thick dashed arrows represent that before entering the gate transformations according to Equations (2.32) have been applied.

2.6.4 ResNet

The ResNet deep learning architecture for image classification was introduced in the paper [17] and it has since been adapted for time series classification which is described in [18, 19]. The ResNet, or residual neural network, is inspired by the *convolutional neural network* (CNN) [20]. The main difference is that the ResNet has "shortcut connections" where inputs can skip a layer in the network. The advantage of this architecture is that this enables deeper networks, which evidence suggest gives better classification results [21]. A CNN is a type of neural network adapted for data in a grid-like form such as images or more importantly in our case, time series. As the name suggests, the CNN involves performing *convolutions* of the inputs in order to extract important features, which is what makes the architecture so successful. The convolution, which is a mathematical operation, can be interpreted as a weighted average where the weights are determined $w(t)$. The operation, in the general continuous case, is given by

$$(f * w)(t) \equiv \int_{-\infty}^{+\infty} f(\tau) w(t - \tau) \, \mathrm{d}\tau \,, \tag{2.33}$$

where f and w are two, not necessarily continuous, real-valued functions. However, if we want to think of the convolution as a weighted average, $w(t)$ must be a valid probability density. For our intents and purposes, the discrete version of the convolution operation of more interest and is given by

$$(f * w)(t) \equiv \sum_{\tau=-\infty}^{+\infty} f(\tau) w(t - \tau). \tag{2.34}$$

For the case of two-dimensional grid-like data with finite structure, the convolution is given by

$$(I * \kappa)(m, n) = (\kappa * I)(m, n) = \sum_j \sum_k I(m - j, n - k) \kappa(j, k), \tag{2.35}$$

where we use the commutative property of the convolution operation. Here κ in equation (2.35) is called the *kernel* or the *filter*, which contains the trainable weights that are updated using backpropagation, similar to the MLP. The input data I is in this case a two-dimensional array of data corresponding to, for example, an image or a multidimensional time series. Data of this form is usually referred to as *tensors* in the context of deep learning and CNNs. Truth be told, the actual mathematical operation implemented in most neural network libraries is actually called *cross-correlation* but still referred to as convolution [10]. When implementing cross-correlation equation (2.35) changes slightly in the following fashion

$$(I * \kappa)(m, n) = (\kappa * I)(m, n) = \sum_j \sum_k I(m + j, n + k) \kappa(j, k). \tag{2.36}$$

Henceforth, we will follow the convention of referring to equation (2.36) as the convolution operation. The kernel κ is an operation that locally extracts features from the data such as edges in images and changes in time series. In order to extract features across the entire input tensor I, the convolution operation is performed

with a sliding window covering I. The size of this sliding window is called the *stride*. The output of convolving one kernel on the entire tensor is called a *feature map* and usually a convolutional layer outputs several of them each with their own corresponding kernel. The size of the kernel and the number of feature maps are tunable hyperparameters. After each convolution, the dimensions of the output will decrease, as we see in Figure 2.8. This is called *valid* convolution. The size reduction of performing the convolution may not be desirable and can be circumvented using *padding*. This action is applied to the input tensor filling the outer borders of the tensor with some value to get the same output size as the original input size. There are different methods but usually one pads the input with zeros. In Figure 2.8 valid convolution is performed to exemplify the convolution operation in the context of CNNs. The basic architecture of the ResNet applied to TSC is shown in Figure 2.9. The ResNet architecture consists of blocks where each of them is made up of convolutional and batch-normalization layers with activation functions (usually ReLU) in series with a certain number of repetitions. Some claim that batch normalization layers help mitigate the internal covariance shift in the network and other say that it also smooths the optimization landscape of the loss function [22,23].

Each block can be bypassed through the short-cut connections. Depending on the amount of data available, the number of blocks in the network can be very large (100 to 1000 blocks) and that kind of depth is enabled by the shortcut connections. The two last layers are the global average pooling (GAP) layer and the softmax layer. The GAP layer calculates the averages of each feature map and returns it to the softmax layer. Finally, the softmax layer outputs the probability over the different classes. The ResNet is optimized by the usual back-propagation algorithm.

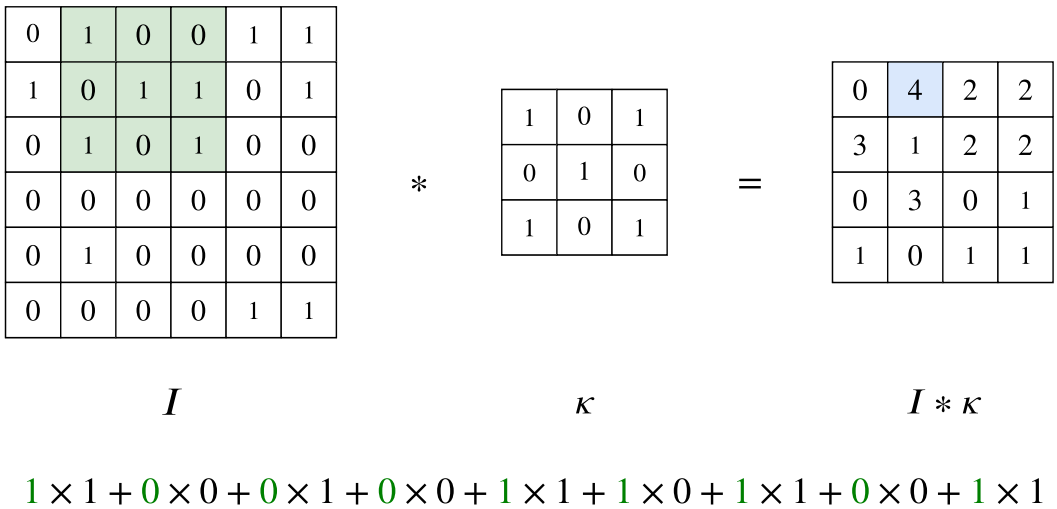


Figure 2.8: Example of "valid" convolution with stride one. The output $I * \kappa$ is called a feature map.

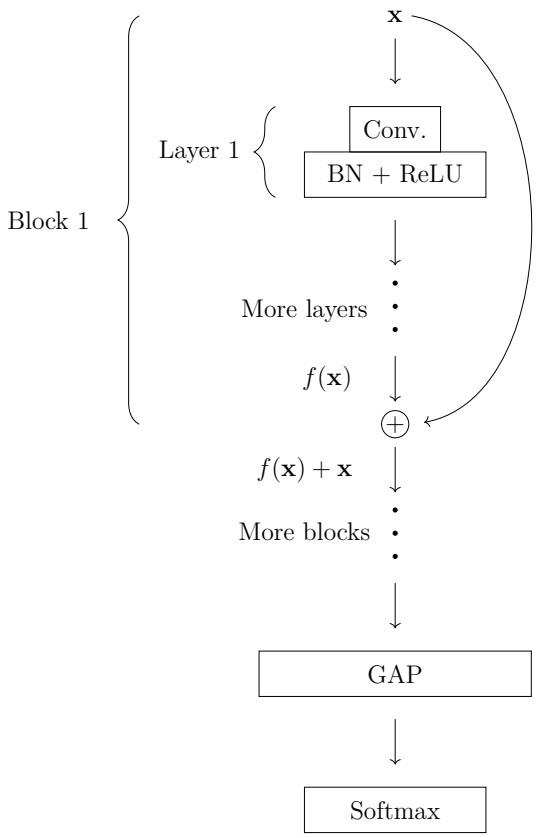


Figure 2.9: The overall structure of the ResNet used for time-series classification. ResNet is characterized by its "short-cut connections", where inputs can bypass the convolutional and batch-normalization/activation layers. The last layers are a GAP layer and lastly a softmax layer.

2.6.5 Time Le-Net

LeNet is an architecture that has a powerful performance for image classification and it was first implemented for document classification in 1998 [18]. The architecture can be observed in Figure 2.11 and it is a traditional CNN with a fully connected layer that performs the task of matching the extracted features to the corresponding class. Finally, the last layer has softmax activation that returns the estimated probabilities for each of the classes. The initial layers are consists of two sets of convolutions layers with ReLU activations, followed by a data downsampling performed by the maxpooling operation. This architecture was first implemented for time series classification in [24].

Maxpooling is a dimension reduction technique applied to the feature maps and it is used to obtain the most essential attributes of the feature maps. The maxpooling layer has some kernel-size and extracts the maximum value in the tensor over that kernel. Similarly to the convolution operation, it has some certain stride which is a hyperparameter chosen by the user. In Figure 2.10 the maxpooling is shown for a two-dimensional tensor.

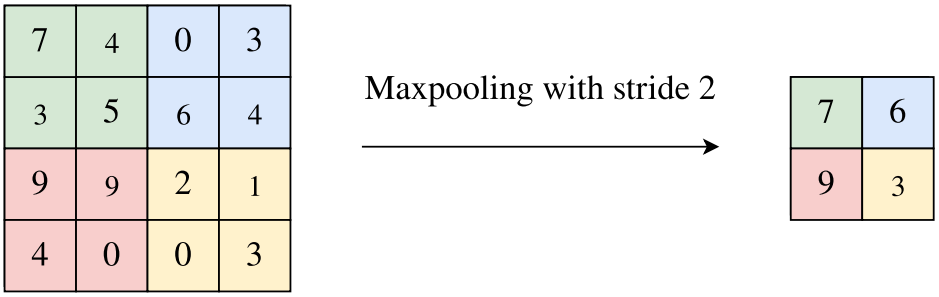


Figure 2.10: The maxpooling operation with stride two on a two-dimensional tensor.

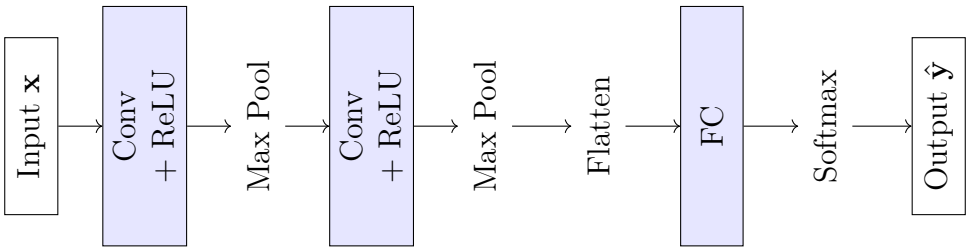


Figure 2.11: LeNet traditional architecture.

3
Pattern Discovery

In this chapter we will describe the methods used for the pattern discovery. The methods are mainly divided into two parts. The first part consists of presenting the steps used for pattern discovery in individual signals and some generalizations of the obtained results. In the second part the patterns found in the signals are combined to describe more complex behaviors. Finally, the results from both parts are used to label the data in meaningful classes.

3.1 Discovering Patterns in Individual Signals

The procedure of categorizing real-valued features will be discussed in this section. This categorization is a necessary step in finding sensible and interpretable multi-dimensional labels corresponding to traffic scenarios. We will refer to categories in the individual signals as patterns. Steering angle and speed signals are the continuous features selected for categorization, since they are important for describing the vehicle state.

3.1.1 General Procedure

In order to find patterns in the individual signals we have developed the following general method:

1. (optional) normalize signal using speed signal to convert time into distance,
2. find changes in the signal and extract sub-sequences of the time series where the changes are found,
3. (optional) process the sub-sequences in some suitable way to prepare for clustering,
4. cluster the sub-sequences to obtain groups of patterns.

The steps in the above procedure changes slightly depending on which signal that is being analyzed. How this procedure can be applied to the speed and the steering wheel angle will be explained in the subsequent sections.

3.1.2 Steering Angle

In this part the general procedure will be applied to the steering wheel angle signal. Each step will be outlined and adapted to this specific signal.

3.1.2.1 Converting time index into distance index

For analyzing and extracting patterns in the steering angle signal it may be advantageous to index the signal with equidistant distance points instead of evenly spaced time points. This is achieved by integrating the speed signal to obtain a mapping from time to distance, $t \mapsto d(t)$. This mapping is defined as follows

$$d(t) = \int_0^t v(\tau)\,\mathrm{d}\tau\,. \tag{3.1}$$

Using the above mapping, the signal previously in time $y(t)$ can be converted into a signal in distance $y(t) \mapsto y(d) = y(d(t))$.

3.1.2.2 Change detection and sub-sequence extraction

Following step 2 in the general procedure, a change detection method is employed in order to extract sub-sequences of the time series. A suitable measure of change for this signal is the first order derivative, meaning that if the derivative is above some certain threshold, a change in the signal is detected. The physical interpretation is that if the angular speed of the steering wheel exceeds some particular value, the driver is performing some action that should be investigated. There is some ambiguity in finding an appropriate threshold for the change detection. Simply choosing a threshold may introduce unwanted bias in the model and may not generalize well, so finding that value automatically is desirable. To understand how this can be done, one must first realize that the number of changes detected N_c is a function of the threshold θ, i. e.,

$$N_c = N_c(\theta), \tag{3.2}$$

where $N_c(0) = N$, $N_c(+\infty) = 0$ and N is the total number of data points in the signal. Each series of consecutive data points where a change was detected is called a sub-sequence. The number of sub-sequences N_s is also a function of the threshold, with $N_s(0) = 1$ and $N_s(+\infty) = 0$. We propose that the threshold should be chosen so that it maximizes the number of sub-sequences found in the signal, that is,

$$\theta^* = \underset{\theta \in [0,\infty)}{\arg\max}\, N_s(\theta). \tag{3.3}$$

This might sound a bit counter-intuitive but the rationale is that if the threshold is chosen as θ^* the most number of sub-sequences are explored which should give more chances to find patterns in the signal.

3.1.2.3 Processing of sub-sequences

Step 3 is an optional processing step where sub-sequences found in the earlier steps may be removed or merged to obtain more coherent and meaningful patterns in the clustering procedure. For the steering wheel signal this step entails removing sub-sequences with low signal amplitude and merging sub-sequences in close proximity. When developing this method we found that the low signal amplitude threshold is

preferably set as the standard deviation of the signal. This statistic for our purposes can be thought of as a measure of the noise in the signal.

Another important step mentioned above is the process of merging sub-sequences close in distance. This step should be automatic to the most possible extent. Our suggestion to automate this step is the following procedure:

i) merge all patterns with an inter-sequence distance lower than some $d_{\min}$,

ii) sort the inter-sequence distances between the sub-sequences d_i, $i = 1, \ldots, N_s - 1$,

iii) find the optimal merging distance d^* threshold in the interval $[d_{\min}, d_{\max}]$, where $d_{\max}$ is the maximum accepted merging distance,

iv) merge sub-sequences with inter-sequence distance lower than or equal to d^* with its closest neighbor.

The optimizing criteria in step iii) for the merging distance is

$$d^* = \underset{i \in \{2,\ldots,N_s\}}{\arg\max} (\delta_i - \delta_{i-1}) i^\alpha, \quad \text{where } \delta_i = f(d_i) - f(d_{i-1}),\ i = 2, \ldots N_s. \tag{3.4}$$

Also $\delta_1 = 0$, $f(d_i) = \sum_{j=1}^{N_s} \mathbb{1}(d_j > d_i)$ (where $\mathbb{1}(d_j > d_i)$ is an indicator function) and α is usually chosen between 0 to 4.

3.1.2.4 Outliers

Before proceeding with the clustering procedure, patterns with deviating characteristics might affect the results. In order to detect these outliers, descriptive statistics such as length, variance, amplitude and skewness (defined in equation (3.7)) of the extracted sub-sequences are calculated and compared using boxplots. Outliers are defined as observations found outside $q_1 - w(q_3 - q_1)$ and $q_3 + w(q_3 - q_1)$, where q_1 and q_3 represent the first and third quartiles, respectively, and w specifies the number of interquartile ranges $(q_3 - q_1)$ above and below q_3 and q_1 respectively. One commonly standard value is $w = 1.5$. Patterns found as outliers more than once in terms of length, variance, amplitude and skewness are not included in the clustering procedure and instead they are labeled as "outliers".

3.1.2.5 Sub-sequence clustering

The last step (4) is to cluster the found sub-sequences to obtain groups corresponding to similar patterns in the signal. Since the extracted sub-sequences have arbitrary lengths, the most suitable way to obtain a pairwise similarity measure is by using DTW. After computing the distance matrix, hierarchical clustering using Ward's minimum variance method is implemented. The number of clusters is specified using the WSS method explained before.

Alternatively, statistical information such as the mean $\bar{y}_s$, the standard deviation S_s, the skewness γ_s and the auto-correlation coefficient $\mathrm{AC}_s^{(\ell)}$ defined as follows

$$\bar{y}_s = \frac{1}{t_{s+1} - t_s} \sum_{i=t_s}^{t_{s+1}-1} y_i = \frac{1}{\tau} \sum_{i=t_s}^{t_{s+1}-1} y_i, \tag{3.5}$$

$$S_s = \left(\frac{1}{\tau + 1} \sum_{i=t_s}^{t_{s+1}-1} (y_i - \bar{y}_s)^2 \right)^{\frac{1}{2}}, \tag{3.6}$$

$$\gamma_s = \frac{\frac{1}{\tau} \sum_{i=t_s}^{t_{s+1}-1} (y_i - \bar{y}_s)^3}{S_s^3}, \tag{3.7}$$

$$\mathrm{AC}_s^{(\ell)} = \frac{\frac{1}{\tau} \sum_{i=t_s}^{t_{s+1}-\ell-1} (y_i - \bar{y}_s)(y_{i+\ell} - \bar{y}_s)}{S_s^2}, \tag{3.8}$$

can be extracted from each sub-sequence and clustered to obtain groups of sub-sequences (patterns). Hierarchical clustering does not have a major impact on the performance, so instead Gaussian mixture model clustering was used to grouped similar patterns. This method is appropriate for the task as it allows for a wider variety in shapes and sizes of the clusters. The main advantage of this procedure is that it has less computational complexity and thus has a shorter running time. DTW has notoriously long computational time ($\mathcal{O}(N^2)$, where N is the number of data points in the longest time series) which is fortunately somewhat mitigated by the sub-sequences generally being quite short with approximately 100 data points on average. In the end we chose to use the DTW clustering procedure. It was found to give a bit more robust results and the method is a bit easier to interpret.

3.1.3 Speed

In this part a description of the pattern finding procedure in section 3.1.1 is given for the particular case of applying it to the speed signal. The word "pattern" may be a bit of a misnomer in this case since we are looking for sections of the signal with particular trends such as increasing or decreasing speed, i. e., changes in the speed corresponding to different levels of acceleration.

One might wonder why those patterns would be interesting when there is already a signal for acceleration given by the acceleration pedal position. One reason that finding these patterns are still worthwhile is because one might think of scenarios where speed is changing independently of the acceleration pedal signal. Likewise, one can also think of scenarios where the speed is decreasing even though the driver is not braking.

For this signal, only step 2 and 4 are necessary to obtain the patterns.

3.1.3.1 Change detection and sub-sequence extraction

In order to find changes in this signal, we propose using change point detection as described in 2.1.2. The idea is to approximate the signal with linear segments. The segmentation is performed so that the total root mean squared error (RMSE) of all the linear segments is minimized. In this method one either predefines the

number of change points (segments) to be detected or the maximum acceptable error (measured in RMSE) of the approximation. We found that setting the number of change points differently for each driving log so that there is some average intensity of change points per minute worked well. In particular, an average rate of ten change points per minute gave reasonable results.

3.1.3.2 Sub-sequence Clustering

The clustering procedure for this signal entails calculating the slopes of the linear approximation segments, representing the acceleration, and then finding suitable levels for categorizing the time series. A suitable clustering algorithm to this end is the Gaussian mixture model.

3.1.4 Results

According to the WSS method for evaluating the clustering goodness-of-fit as described in Section 2.3.3, a suitable number of clusters for grouping the sub-sequences is three clusters, as can be seen in Figure 3.1. Figure 3.2 shows the grouping results from one of the driving logs. The sub-sequences plotted in blue, red and green mostly correspond to roundabouts, turning right and turning left road patterns, respectively. By using the criteria to identify patterns with deviating characteristics, (section 3.1.2.4) 30 outliers were identified, corresponding to approximately one or two patterns in each of the log files. The sub-sequences plotted in black are the ones categorized as an outlier and they mostly correspond to a situation where the vehicle is parking. Data points not grouped in any of the mentioned patterns will be referred to as the "no turning" or "driving straight" situation. As its name suggests, this pattern corresponds mostly when the vehicle is driving straight.

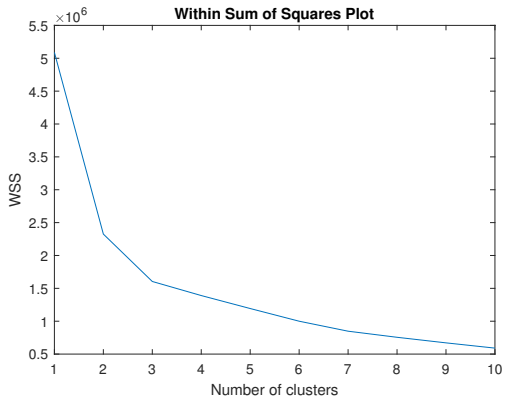


Figure 3.1: Elbow method performed to identify a good fit for grouping the different shapes found in the steering wheel angle signal. The maximum bend in the plot corresponds to three clusters.

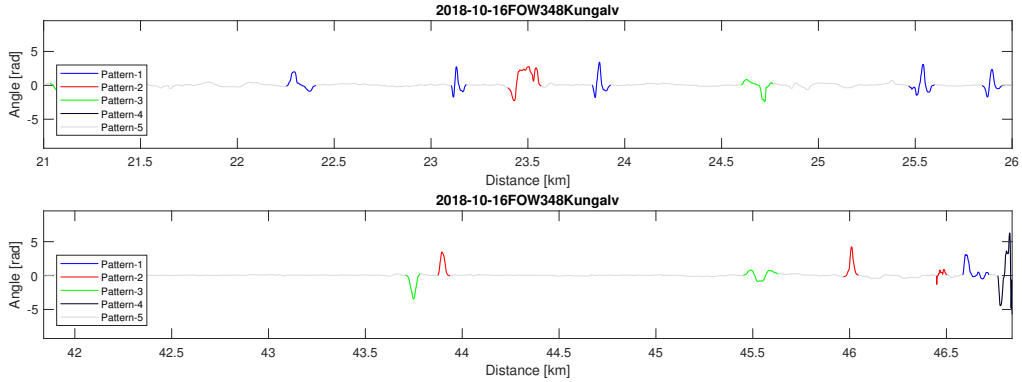
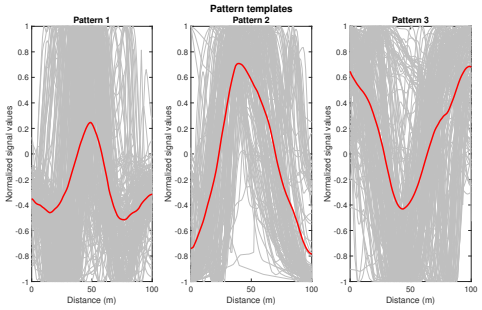


Figure 3.2: Patterns found after clustering procedure of steering wheel angle sub-sequences. Blue, red and green sub-sequences mostly correspond to roundabouts, right and left turns, respectively. Patterns in black correspond to parking maneuvers and they were identified as outliers. Data points not in any of the previous patterns are addressed as a "steady" behavior and correspond to the vehicle driving straight.

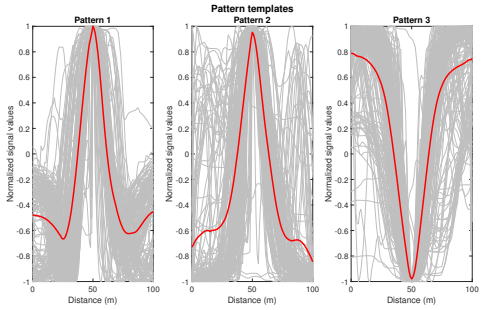
Figure 3.3: Some of the patterns in Figure 3.2 showed on a map.

It is also of interest to analyze the average shape of each of the discovered patterns. The average shape of a particular group can be interpreted as its template and it can help to understand the discovered driver behavior. In Figure 3.4 the templates obtained from the steering wheel angle signal are shown. The templates in Figure 3.4a are found by first normalizing the signal values for each pattern so that they lie within the interval $[-1, 1]$. The patterns are of different lengths so an additional mapping of the sample points onto the interval $[0, 100]$ is performed. Pattern templates in Figure 3.4b are obtained differently from the previously mentioned. This

involves initially aligning the peak or valley in the center of the shapes and then the feature average is calculated some distance around the center. This procedure shows clearer results and it allows to see a symmetry in the driving behavior.



(a) Pattern templates obtained by normalizing the signal values of each pattern and interpolating each pattern to fit on the $[0, 100]$ interval. The red line corresponds to the mean of all patterns.



(b) Peak-aligned pattern templates obtained by normalizing the signal values of each pattern and showing the pattern on a symmetric interval where the peak is set in the mid-point. The red line corresponds to the mean of the aligned patterns.

Figure 3.4: Templates of the obtained patterns in the steering wheel angle signal.

3.1.4.1 Verification of Steering Wheel Angle Patterns

The patterns in Figure 3.2 can be inspected in the driving route map data in order to verify the performance of the pattern discovery algorithm for the steering wheel angle signal. In Figure 3.3 some of the patterns are shown on the map.

Utilizing that all driving logs are recorded on the same route the verification of the steering wheel angle patterns can be done using the GPS-data when it is available. The rationale is that since intersections (left/right turns and roundabouts) are located at the same place in each driving log, patterns found by the algorithm close to those geographical points should be explained by the type of junction found there. So if a pattern is found close to a site known to have a roundabout, we would expect a well performing algorithm to recognize it as a roundabout pattern. In Figure 3.5 the verification procedure is exemplified by plotting the patterns discovered on the map. The total number of driving logs with GPS-data available were 14 of the total 22 logs. In the logs with GPS-data there is a total of 317 patterns and out of those 265 were close to a geographical ground truth site. 204 patterns agreed with the ground truth site, i.e., the intersection pattern category as decided by the algorithm agreed with the actual intersection type giving a total of 77 % agreement. In Table 3.1 the confusion matrix of the patterns discovered and the actual intersections are shown.

One might wonder, if the ground truth is available in some cases, why do we not just use that labeling instead? This question has a few different answers. First, not all driving logs have the GPS data available and just using the ones that have would resulted in a serious reduction of the available data. Secondly, what we set

out to do in this thesis is to *discover* driving scenarios. Sure, if the goal of the thesis was to find different types of junctions, then this is probably not the most efficient or best procedure at all. What we have done here is that we found some different ways of segmenting the time series given and then we let those segments structure themselves according to similarities between segments. It just so happened that clusters obtained corresponded mostly to different types of intersections but that was never intended in the first place. Finally, there might be an underlying reason for the grouping obtained that is not evident at present. Maybe there is some perspective that we do not know from which it make sense that some left turns are labeled as roundabouts and vice versa. This is always the problem with unsupervised learning - it is not always evident for an outside observer why the results are what they are.

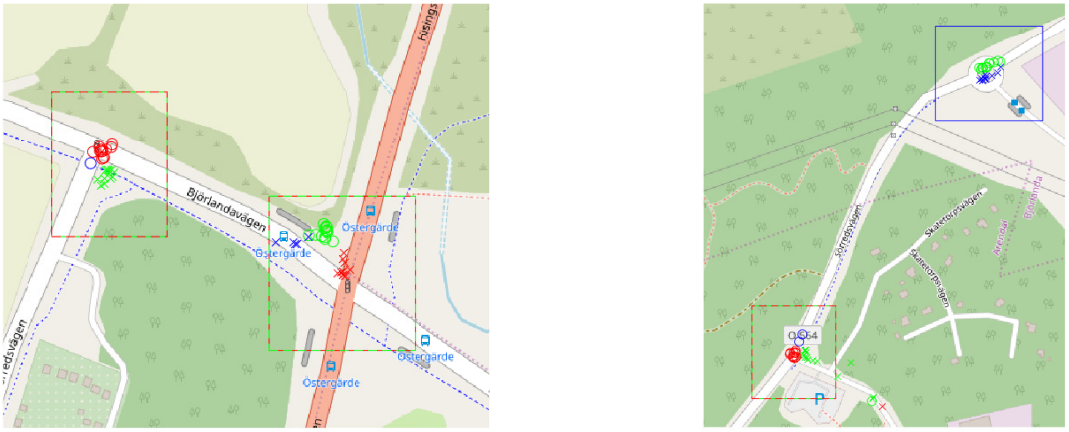


Figure 3.5: The verification procedure of the steering wheel angle pattern discovery. The 'x'-markings indicate that the vehicle is moving from Volvo Torslanda towards Kungälv and 'o'-markings mean traveling in the opposite direction. The rectangles mark geographical sites with known intersection types. The green and red dashed rectangles indicate left and right turns (depending on travel direction) and blue rectangles indicate roundabout sites. Likewise, markings that are red or green signify that the algorithm recognized a left or right turn, respectively, and blue markings denote finding a roundabout pattern.

3.1.4.2 Intersection behaviors

Now that a method for correctly identifying the different junctions have been developed, we can analyze common behaviors and actions at various types of intersections. In order to draw reliable conclusions about the driver behavior in intersections we do not use the labels from the pattern discovery algorithm since about 23 % of those labels do not agree with the ground truth. Instead we relabel the patterns according to the ground truth classification.

		Detected		
		Roundabout	Left	Right
True	Roundabout	82	16	24
	Left	19	52	1
	Right	0	1	70
Total		122	72	71

Table 3.1: The confusion matrix of the actual intersections and the patterns the algorithm found. The overall agreement is 77 %.

Studying the driver's behavior and actions means analyzing the other features when passing through the different intersections, such as power demand, acceleration pedal position, speed and braking. This is achieved by aligning the peaks of each steering wheel angle pattern, like it was done for obtaining the templates in Figure 3.4b. Then the feature behaviour is analyzed at a certain distance from the center of the pattern.

In Figure 3.7 we see the driver behaviors while entering the different types of junctions. The analyzed signals are the power demand, acceleration pedal position and the vehicle speed. New pattern templates are created, as seen in Figure 3.7a, where each corresponds to an actual intersection such as left/right turn and roundabout. To note is that the templates remain more or less unchanged as compared to 3.4b. However, we can see that there is more variance in the patterns corresponding to roundabouts whereas the patterns corresponding to left and right turns have less variation. When visually analyzing the different driver actions at the junctions, the median feature value for each intersection type is plotted with a 50 % envelope. If the medians are all contained in the envelope-overlap it indicates that the driver behaviors are similar between the different junction types. One can observe that most driver actions depend on the type of intersection and only the vehicle speed seemed to be roughly the same for all junction types. In order to rigorously test whether the behaviors are different in the various types of junctions, one can perform a statistical test, such as the *Kruskal-Wallis test* on the mean distributions for each type of intersection shown in Figure 3.6. The Kruskal-Wallis test is a type of non-parametric ANOVA (ANalysis Of Variance) test. The null-hypothesis H_0 states that the distribution of all k samples are equal and the alternative hypothesis H_1 asserts that at least one of the samples dominate stochastically [25], which can be formalized in the following way

$$\begin{aligned} H_0 &: \quad M_1 = M_2 = \cdots = M_k, \\ H_1 &: \quad \exists\, i, j \in \{1, \ldots, k\} :\ M_i \neq M_j, \end{aligned} \tag{3.9}$$

where M_i is the median of the ith sample distribution. In our case $k = 3$ representing the different types of intersections. The Kruskal-Wallis test does not tell which sample distribution is stochastically dominant, only that there is at least one. Fortunately, there is a post hoc test called Dunn's test which is performed after the Kruskal-Wallis test, to determine which samples have statistically significantly dif-

ferent distributions [26]. In Table 3.2 the results from the statistical tests are shown. As it was indicated before, there is no significant difference in driver actions when it comes to vehicle speed. For the other features, a statistically significant difference in behavior was found. In particular, there seems to be a significant difference between the behaviors in roundabouts and left turns. It is also interesting to note that no statistical difference between the roundabout and the right turn was observed. One explanation for this is that a roundabout involves an initial right turn which may be why these situations do not differ as much in terms of driver behavior.

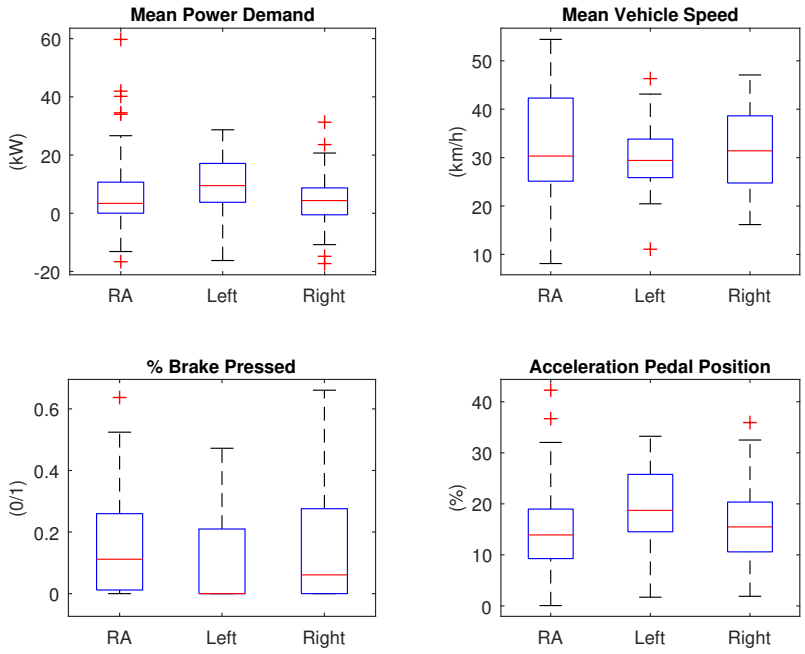
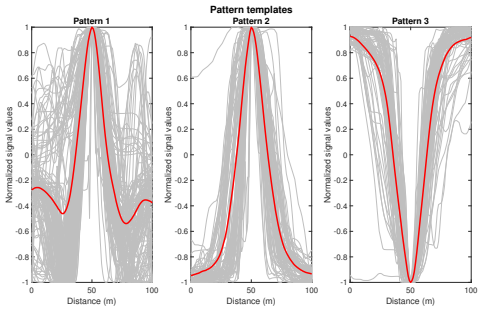


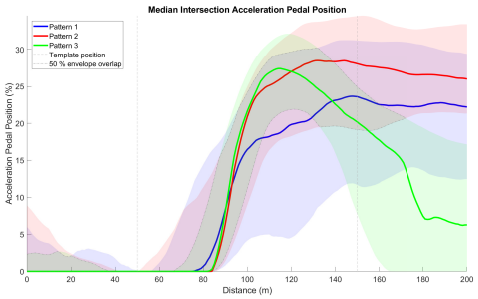
Figure 3.6: The mean distributions for each feature considered in each type of intersection.

Feature	**K-W test** p**-value**	**Sign. dev. (Dunn's test)**
Power Demand	0.00153	RA, left Left, right
Speed	0.354	NA
% **Braking**	0.0308	RA, left
Acc. ped. pos.	<0.001	RA,left Left, right

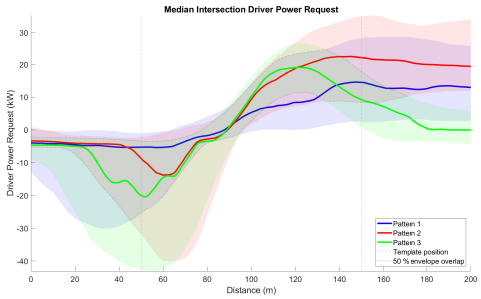
Table 3.2: The results for the different statistical tests performed on the distributions in Figure 3.6. The left column shows the different features considered. The middle column shows the p-value when carrying out the Kruskal-Wallis test. The right column shows which intersections deviate significantly in terms of the features at the significance level $p = .05$.



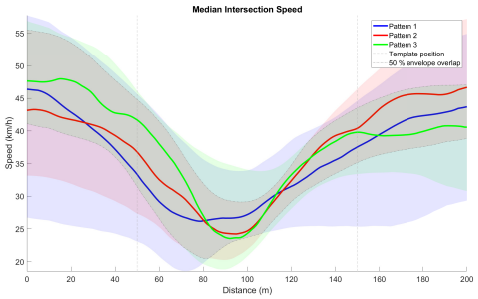
(a) Steering wheel angle templates from patterns agreeing with the ground truth intersections. From left to right, the plots correspond to the roundabout, left turn and right turn intersections, respectively.



(b) Median acceleration pattern profile. The signal takes percentile values which corresponds to how much the acceleration pedal is pressed. We see that the behaviors are similar when entering the intersection and diverges after leaving the intersection.



(c) Median power demand patterns profile. The behaviors are similar before entering the intersection and the behaviors diverges as the driver leaves the roundabout.



(d) Median speed pattern profile. Here all behaviors are contained within the envelope overlap which indicates that driver behaviors and actions do not depend on the type of intersection when it comes to vehicle speed.

Figure 3.7: In the upper left figure the pattern templates from the actual intersection types are shown. In the upper right and the two bottom figures driver behaviors at different distances in the intersections are shown. In particular, the median power demand, acceleration pedal position and vehicle speed are displayed. In addition the 50 % envelope is plotted in the corresponding color. The envelope overlap is shown in a grayish transparent color. The gray dashed lines corresponds to where the steering wheel angle template is situated. The colors blue, red and green represent roundabout, left and right turn intersections, respectively.

3.2 Discovering Pattern Sequences

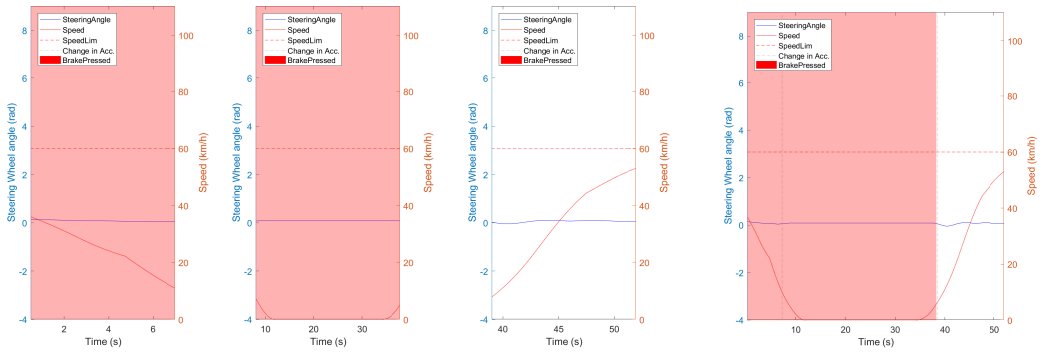
Now that the real valued features have been labeled as described above, the next step is to combine these results to obtain multidimensional labels of the data. This is done by combining patterns from the different features happening at the same time to obtain more specific labels and by looking at repetitive groups of labels

happening in sequence. In this section we will explain this approach in detail.

3.2.1 Finding Traffic Scenarios

Since the continuous features have now been labeled according to the type of pattern, a unique Behavior Symbol (BS) can be assigned to each combination of labels in the features, just like in step ii in section 2.4. From now on, this unique BS will be called a traffic situation and some examples can be observed in Figures 3.8a, 3.8b and 3.8c. Consecutive BSs are identified to obtain the modified BS sequence (step iii in section 2.4). Then, a suitable window size w_{BS} is defined to find an interpretative combination of BS patterns. The results of this procedure is what is called a traffic scenario (Figure 3.8d). Instead of using the MDL principle criteria to choose the optimal motif in the data, the cost function will be based on the number of repetitions of the pattern and it is independent of its duration. In this case, we are looking for the maximum cost function, that is, the most repeated traffic scenarios.

For the discovery of traffic scenarios, a window size from 1 to 3 has been chosen and the 10 most repeated scenarios are analyzed. These scenarios are listed from 5 to 10 in Table A.1 in the Appendix, as well as some information about their duration and number of repetitions in the data set. Even though we are analyzing a specific route, the locations of the scenarios might change every time since they also depend on the interaction of the vehicle with the current traffic.



(a) BS B: Decelerating and braking

(b) BS C: Constant speed and braking.

(c) BS D: Accelerating.

(d) Traffic scenario composed by the consecutive Behavior Symbols BCD.

Figure 3.8: Example of a traffic scenario obtained from sequential Behavior Symbols.

3.2.2 Labeling Data

Our data set can be now labeled using the classes obtained from the steering wheel angle signal and using the scenarios obtained from pattern sequences. The labeling is interactive in the sense that suggestions for traffic scenarios are shown and then the user chooses the ones that are suitable for labeling the data. In order to avoid

labeling data points more than once, classes obtained from the steering wheel angle signal are labeled first. Then, the unlabeled data are used in the pattern discovery algorithm to obtain the most repeated combinations of BSs. After showing the scenario with the maximum cost function, the user is able to save it or look at the next one. After choosing a scenario, the data points in it are cancelled out for the next run and the algorithm is repeated again to find the next most repeated combination with the remaining unlabeled data.The algorithm stops after finding the specified number of scenarios.

To be able to label additional data sets later on from other routes, the segmentation procedure described in Sections 3.1.2.1, 3.1.2.2 and 3.1.2.3 can be repeated to obtain a segmentation of the steering angle signal. The already classified steering wheel angle patterns from the Kungälv route can be then used as a set for training a 1-Nearest Neighbor (1-NN) classifier to predict labels in new segments using the DTW distance. These labels are based on the proximity of the patterns in the training data. Despite its simplicity, 1-NN is considered a powerful approach for Time Series classification (TSC) [27]. The distribution of the descriptive statistics (subsection 3.1.2.4) on the training set can be used to identify outliers by using the same criteria.

3.2.3 Results

The scenarios obtained from the steering wheel angle signal (1-4) and the pattern discovery algorithm (5-10) are listed in Table A.1 in the Appendix. A description for each scenario and some information such as the number of repetitions and duration proportion in the data set are also included. The most repeated scenario is the called “steady” or “driving straight” and here the vehicle keeps a constant speed without braking or turning. This state is commonly found when driving on the highway and its duration can vary from five seconds to four minutes in the data set. This duration is usually longer if the vehicle drives on the highway for a long period of time. It is also identified as the scenario that represents 54 % of the data. The second biggest class is the roundabout junction. This one is obtained from analyzing the steering wheel angle signal and it is one of the three analyzed intersections in a previous section. It covers approximately 5 % of the data and it has a median duration of 23 seconds. Another of the scenarios found is the one referred to as “waiting in traffic”. There, the vehicle initially decelerates while braking. Then it stops for a few seconds and then accelerates again. This behavior is commonly found when stopping at a traffic light or waiting in a queue. Its duration in our data set seems to variate between nine second and almost two minutes.

4
Classification of Driving Scenarios

In the previous chapter methods for pattern discovery were implemented to label the driving data. In this chapter we will build classification models to predict the current traffic scenario using the driving features. The section will cover different methodologies and neural network architectures for reaching the goal.

4.1 Data Preparation

One of the goals of this thesis is to investigate the feasibility of an online driving scenario classifier. To that end, an experimental setup simulating the online data recording is needed. In an online classification situation data would arrive to the classifier in a stream while driving along the route. At certain time intervals the classifier would make a classification of the current driving situation by analyzing the inputs from a few seconds ago to the current time.

This situation can be simulated using the driving data set by slicing it into chunks of time length t_s and classify the corresponding label at the end of each data block. This means that if the original data $\mathbf{x}$ is a matrix of size $p \times N$ (p features and N observations), the new data frame $\mathbf{X}$ will have $N/t_w \equiv n$ observations of time sequences of length t_s where t_w is the window size of the stream of data chunks, i.e., the time interval between each time sequence. To put this into equations, we have

$$X_i = [\mathbf{x}_{(i-1)\times t_w+1}, \ldots, \mathbf{x}_{(i-1)\times t_w+t_s}], \quad i = 1, \ldots, n, \tag{4.1}$$

where

$$\mathbf{x}_j = \begin{bmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jp} \end{bmatrix}, \quad j = 1, \ldots, N. \tag{4.2}$$

The corresponding new labels $\mathbf{Y}$ are given by

$$Y_i = y_{(i-1)\times t_w+t_s}. \tag{4.3}$$

Now the previous labeled data set $(\mathbf{x}, \mathbf{y})$ has been transformed into a set of labeled time sequences $(\mathbf{X}, \mathbf{Y})$ suitable for training a classifier. In Figure 4.1 the process of the data set preparation $(\mathbf{X}, \mathbf{Y})$ is described in a diagram. For the scenario classification, we will focus on training the model to identify the classes obtained from the steering wheel angle signal, which corresponds to the three types of intersections

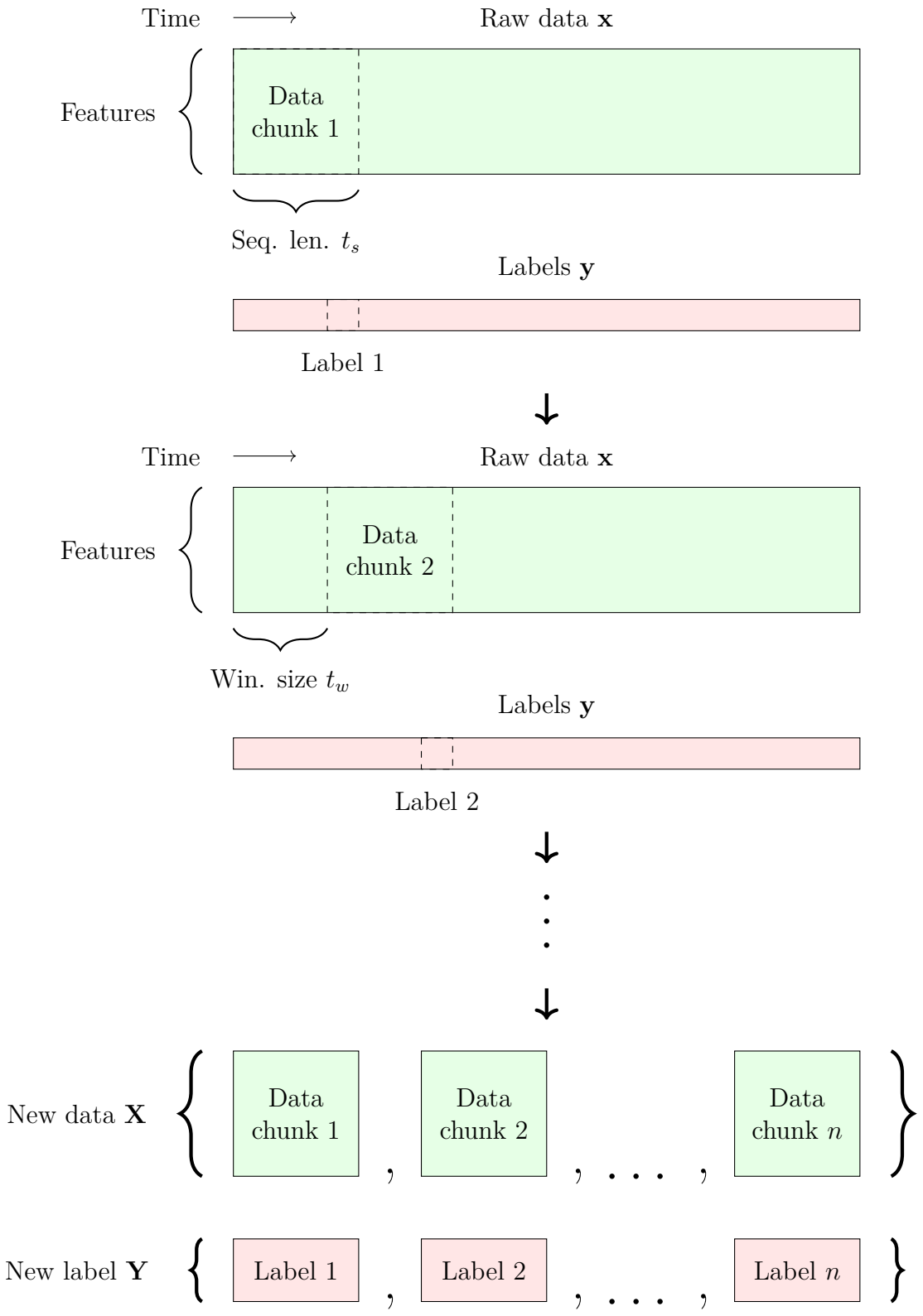


Figure 4.1: A description of how data is prepared for the classification task.

listed in (3.2.3), the "outlier" label corresponding to driving slowly in a parking lot and the remaining time points are labeled as "driving straight" or "steady state".

The features used to identify the labels are the five driving maneuvers mentioned before. As it also mentioned, our data set is composed by 22 log files. The data frame for the classification task is composed by $N = 290,196$ time series data points recorded with a frequency of 4 Hz. This data is split into 80 % for training, 10 % for validation and 10 % for testing. When feeding the data into the classification models, it is downsampled to 1 Hz and each input block of time series information has a length of $t_s = 10$, corresponding to 10 seconds of historical data. The window size has been set to $t_w = 1$, meaning that two consecutive inputs into the networks have an overlap of 9 seconds.

As mentioned before, the labels in the data are dominated by one class, namely the one where no turning is happening and the driver is driving straight. In Table 4.1 the count and proportion of the different classes in the training data is shown.

Label	**No turning**	**RA**	**Left**	**Right**	**Outlier**	**Total**
Count	50053	3193	1687	1988	1119	58040
%	86.239	5.501	2.907	3.425	1.928	100

Table 4.1: Breakdown of the data into the different classes. Note that the largest class is dominating with approximately 86 %.

4.2 Model Architectures

In order to accomplish the classification task, four different network architectures have been chosen due to their success in time series classification and we wish to compare their performances on our problem. The architectures are t-leNet, ResNet, LSTM and stacked LSTMs.

The t-LeNet architecture is proposed in [24]. It is composed by two convolution layers with five and twenty filters respectively, each followed by a one-dimensional max-pooling downsampling operation. The first one has a kernel size of $[2 \times 1]$ and the last max-pooling layer has a filter size of $[4 \times 1]$. For the fully connected layer, 500 neurons were used and this one is followed by a softmax layer. In the paper, data augmentation techniques were applied to increase the size of the data set. This part was omitted in our implementation.

The LSTM architecture is a recurrent neural network composed by one hidden layer with 64 neurons and ReLU activation, followed by a fully connected layer with softmax activation to obtain class probabilities for each outcome.

Stacked LSTMs, as its name suggests, is a deeper network with several stacked hidden LSTM layers. Deeper designs promise to enhance the performance of the network [28]. Its architecture is composed by 3 LSTM layers with 16 neurons each followed by a fully connected layer with softmax activation.

The ResNet architecture consists of four blocks which each have three hidden convolutional layers, as described in section 2.6.4. The three convolutional layers have a kernel size of 8×1, 5×1 and 3×1 respectively. The convolutional layers of the first block outputs eight feature maps each while the last three blocks have convolutional layers with 16 filters. Table 4.2 shows the optimization hyperparameters

for the the four implemented classification models.

ReduceLR is a reference to the function ReduceLROnPlateau from the Keras library. The purpose of this function is to decrease the learning rate when the specified metric has stopped improving after certain number of epochs [29]. This practice helps reducing the risk of getting stuck in a local minimum. The function has been set to have a patience of 10 epochs and if the metric does not improve, the learning rate is decreased by some fraction. In order to deal with the large imbalance of class sizes, the loss function used for most of the models is the focal loss with the parameters α and γ specified in Table 4.2.

Model	Optim.	Loss	γ	α	Decay	LR	Epochs	Batch
t-LeNet	Adam	FL	1	0.25	ReduceLR	0.01	100	256
ResNet	Nadam	CE	NA	NA	0.004	0.002	100	128
LSTM	Adam	FL	1	0.25	ReduceLR	0.001	100	128
S-LSTM	Adam	FL	5	0.25	ReduceLR	0.001	100	128

Table 4.2: Optimization hyperparameters of the compared classification models.

4.3 Results

In this part of the thesis we summarize the results from the classification process. In Figure 4.2 the training performance over time for each of the models can be visualized. The solid line shows the training accuracy, while the dashed line shows the accuracy obtained on the validation data. S-LSTM model seems to have a gradual improvement of accuracy through every epoch and it also takes longer to reach the stopping criteria than the rest of the models. t-LeNet seems to overfit at an earlier stage than the other models and finally ResNet seems to reach its maximum accuracy in the early stage of the training process.

Since the final layer in the model architectures has softmax activation, the output of each model is the probability of belonging to any of the five classes. An example of this result can be observed in the upper plot in Figure 4.3. Here a section of the route is classified using the ResNet model and the probabilities for each class is plotted with a different color for each time point. Finally, the labels are assigned by choosing the class with the maximum probability, see equation (2.24), as it can be observed in the lower plot.

The classification models are trained to identify five different classes: roundabout, left turn, right turn, outliers and no turning/driving straight. The first three classes correspond to different junctions, the fourth category is mostly found when driving in parking lots and the last group is simply when the vehicle is driving straight. Not knowing the current state of the vehicle can be represented by a "model" where all the predictions are the driving straight class. This is the current situation in the vehicle system since no traffic scenario detection is implemented. Let us call this situation the "no model" classification and it is going to work as a baseline for comparing the four architectures. The usefulness of a classifier is questionable if it systematically fails to outperform the "no model" classification.

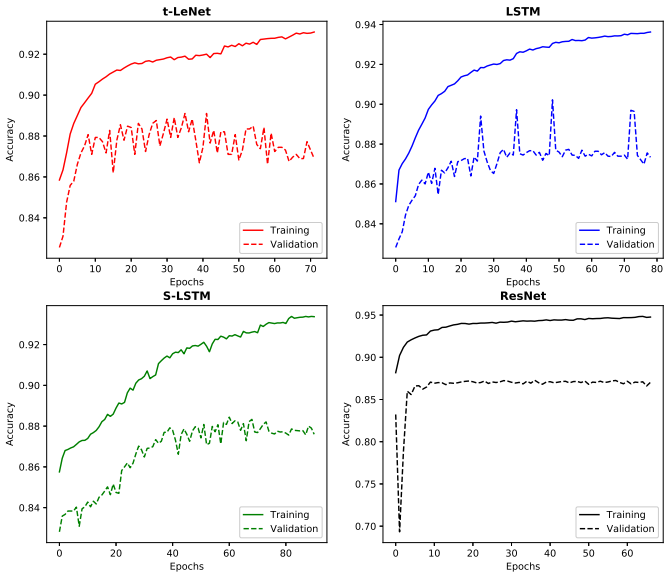


Figure 4.2: Accuracy performance for each of the models during training. The number of epochs before reaching the early stopping criteria is different for each model. The solid line shows the training accuracy and the dashed line the accuracy obtained using the validation data.

A general performance of the different classification models can be observed in Table 4.4. For this comparison, the models outcome has been classified as intersection (positive case) and driving straight (negative case). As it was mentioned before, most of the data points belong to the driving straight. Guessing that all of the data belongs to this class (no model classification) achieves a test accuracy of 82 %. In order to make a fair measurement of the model's performance, we can use other evaluation metrics such as recall, precision and F1 score. The first one can describe the model's ability to detect intersections, while precision is a measurement of

its ability to correctly identify junctions. A good classifier must have both properties, which is why F1 score is also considered for measuring performance.

In Table 4.3 we can observe the four possible outcomes that can be obtained from the classification models. Identifying the current intersection (true positive) makes possible to optimize some of the routines in the vehicle control system such as gear shifting and changes in the propulsion system. Failing to detect to be in an intersection (false negative) leads to the current not optimal system which corresponds to predicting that the vehicle is "driving straight" the whole time. However, predicting to be in an intersection when it is not the case (false positive) can cause a pessimal vehicle control system decision by making non-necessary changes. Favoring results where FP is low implies that we should aim for a high precision rate in our models.

As it can be observed from Table 4.4, t-LeNet classification seems to outperform the rest of the models in terms of precision, but its recall, similarly to the rest of the models, is quite low. Together with Stacked LSTM, these models seem to have

the largest trade-off between precision and recall, making them the most suitable for identifying the driving situation.

		Predicted	
		Yes	No
Actual	Yes	Correctly identifying an intersection (TP)	Failing to detect an intersection (FN)
	No	Erroneously detecting an intersection (FP)	Correctly identifying driving straight (TN)

Table 4.3: Confusion matrix for intersection identification. The desired performance is a high TP rate (green) and low FP rate (red).

Models	**Performance**					
	Acc.	**Recall**	**Precision**	**F1**	**# Epochs**	**# Params.**
t-LeNet	**0.9372**	0.6829	**0.9451**	0.7929	71	13,665
ResNet	0.9177	0.6145	0.8818	0.7243	67	18,369
LSTM	0.9285	0.6668	0.9011	0.7665	79	18,245
S-LSTM	0.9364	**0.6925**	0.9281	**0.7932**	91	5,717
No Model	0.8202	0	NA	NA	NA	NA

Table 4.4: General performance of the models for classifying steering wheel angle patterns.

In terms of accuracy and precision, the models perform better than the baseline. All models have a relatively low recall which indicates that there is a high rate of FN predictions. This means that the models are struggling to detect intersections in some of the cases. Table 4.5 and 4.6 shows the ability of each of the models to identify individuals classes. By looking at the precision table, t-LeNet outperforms the rest of the models in right turn and outlier identification, while ResNet is better at identifying roundabouts. It is worth to mention that this class is one of the most difficult to identify according to the patterns verification since this junction comes in different shapes and sizes. Finally, Stacked LSTM obtained the highest rate for left class identification.

To measure how well the models identify when the car is going straight, lets define this class as the positive case, while the negative one is represented as the vehicle driving in an intersection. Then, predicting to drive straight when it is not the case (false positive) leads to decision making in accordance with the current control system, but failing to detect going straight (false negative) can lead to non optimal decisions. Therefore we can infer that the model should also aim for a high recall for the driving straight prediction. If this is the case, then the "no model" classifier is naturally the strongest, but it is also trivial. The next one best is t-LeNet, with almost 99 % recall and the stacked LSTMs is a close third. Saying this, we can then conclude that t-LeNet architecture has the better performance by having the best identification of three out of five classes.

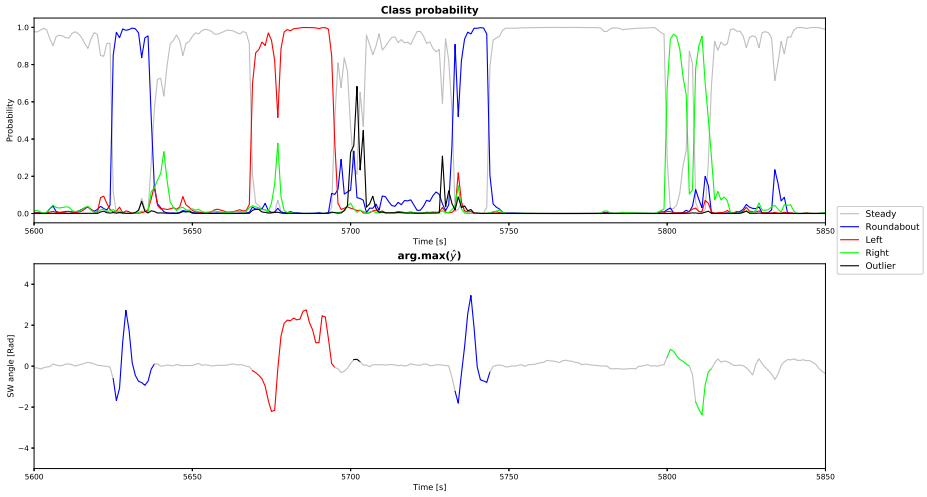


Figure 4.3: Classification output obtained using ResNet model. The upper plot shows the probability for each of the classes. The lower plot shows the signal colored with the label with maximum probability.

Models	Precision by class				
	Steady	**RA**	**Left**	**Right**	**Outlier**
t-LeNet	0.9335	0.5069	0.6147	**0.8316**	**0.9204**
ResNet	0.9305	**0.6003**	0.5721	0.5838	0.6611
LSTM	0.9310	0.2422	0.6705	0.7567	0.8449
S-LSTM	**0.9336**	0.2470	**0.6931**	0.7299	0.7857
No model	0.8202	NA	NA	NA	NA

Table 4.5: Precision performance for identifying steering wheel angle classes using each of the selected models.

Models	Recall by class				
	Steady	**RA**	**Left**	**Right**	**Outlier**
t-LeNet	**0.9893**	0.4602	0.4076	**0.5121**	**0.6448**
ResNet	0.9864	**0.5065**	**0.5129**	0.2719	0.4974
LSTM	0.9838	0.4664	0.4773	0.4705	0.1967
S-LSTM	0.9870	0.4701	0.5020	0.4201	0.1985
No model	1	0	0	0	0

Table 4.6: Recall performance for identifying steering wheel angle classes using each of the selected models.

Outlier precision has a general average of 80%, making it the easiest type of scenario to predict (besides the "driving straight scenario"). It is followed by turning right, with 72 %, then left turn with 63 % and finally roundabout with 40 %.

In Figure 4.4, a small route section is shown and the classification made by each of the models is marked with different colors. In this example, stacked LSTM seems to disagree more with the rest of the classifiers about the intersections by making more right turn predictions and alternating between the classes more frequently than the rest. From this figure it can also be seen that there is a large agreement between the algorithms for classifying the roundabout label for this specific section.

In order to prove the robustness of the models, we attempt to label a new route section in Karlstad, which is completely unseen driving data from a different location than the route in the training data. Since we do not have the labels available, the performance comparison can only be made visually. In Figure 4.5 we see the classification made by the four models. LSTM and ResNet seem to have a good identification of the "driving straight" case. On the other hand, they also show problems when it comes to differentiating between right turns and roundabouts. t-Lenet and stacked LSTM models seem to more easily identify roundabouts and right turns, but still they seem to have the problem of frequently switching between different classes while driving in the intersection.

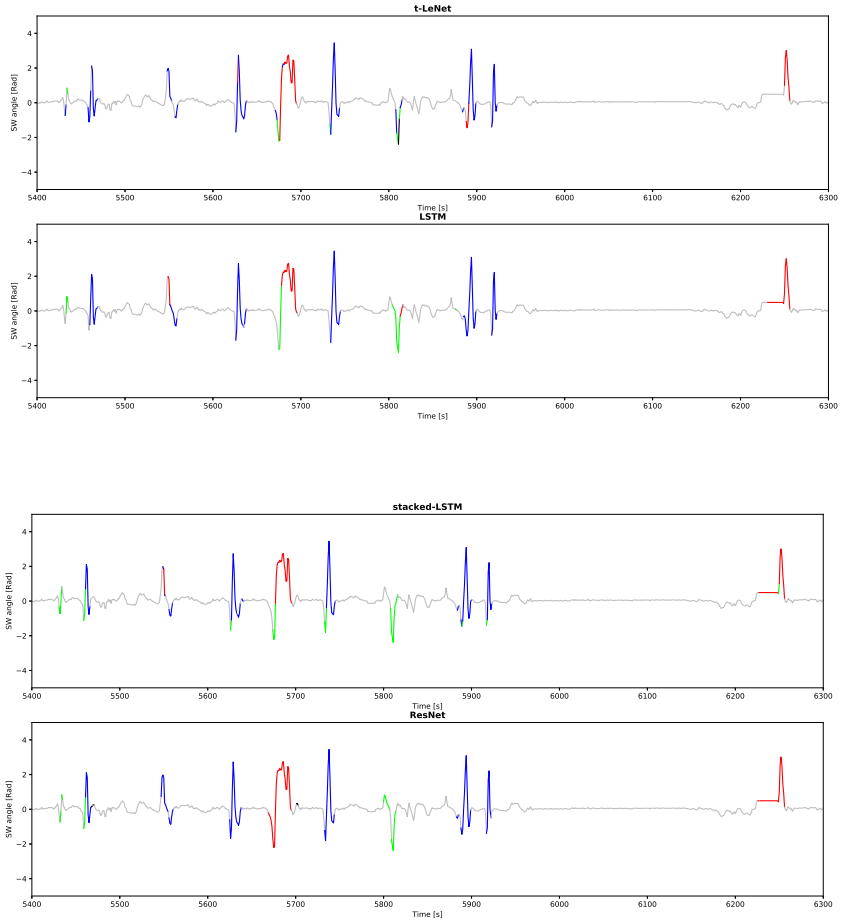


Figure 4.4: Data classification of one section route of the test data using the four classification models. Roundabout classifications are colored in blue, left turns in red and right turns in green.

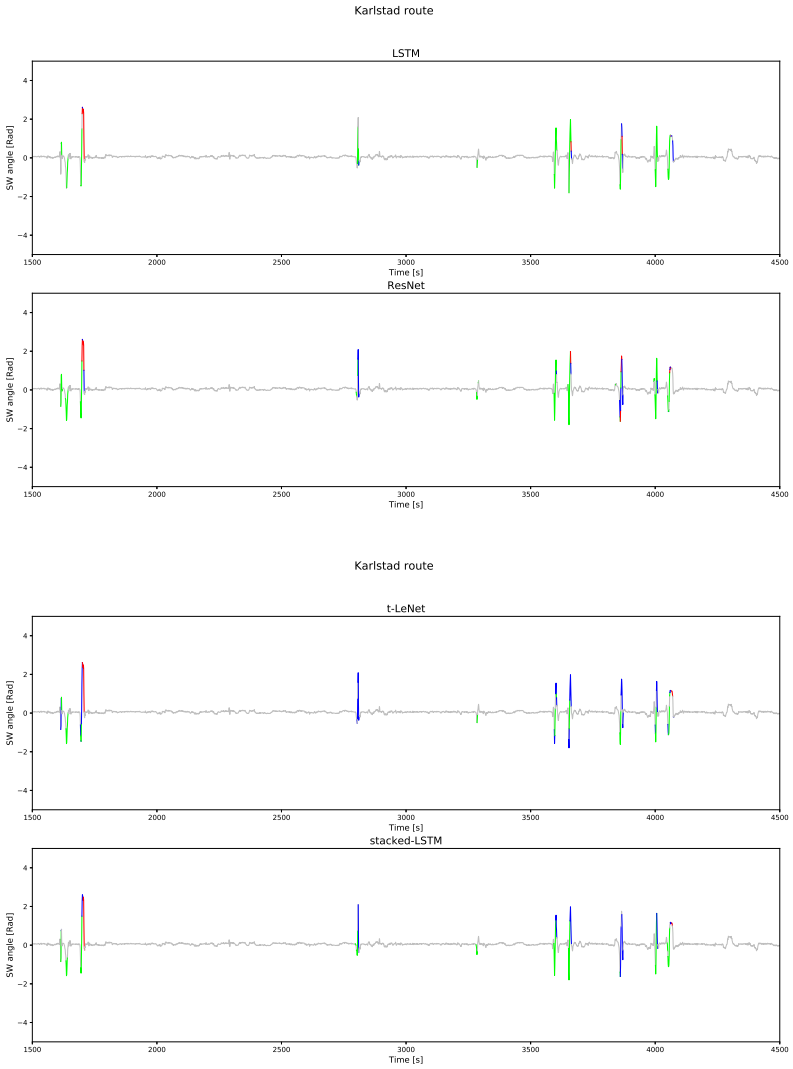


Figure 4.5: Data classification of one hour route in Karlstad using the four networks. Roundabout classifications are colored in blue, left turns in red and right turns in green.

5
Conclusion

In this chapter we discuss the results in the previous part, the methodology used and the implications of the work done in this thesis for future and related research.

5.1 Summary

In this section we will summarize the methodology used in this thesis and the results that were produced.

5.1.1 Methodology

As have been mentioned before, this thesis consists of two main parts - discovery and labeling of driving scenarios and classifying those scenarios in new driving data. To this end, the first part of the thesis consisted of employing various unsupervised learning and data mining techniques to obtain labels to train a classifier on. One of our goals in the first part is to design an automated discovering process free of preconceptions such as defining what constitutes a common driving scenario. In this way one might find patterns that was not expected and truly discover something new about the data. However, in this process we realized that some element of human interaction in the labeling process is need if one wishes to obtain results that are interpretable. Some time was spent in the early stages of this thesis to cluster entire driving routes, just based on some information about each time segment in order to achieve a labeling based on clustering classes of those segments. The problem with this approach is that the groups of "driving scenarios" that are discovered through this process is hard to interpret and verify.

This led us to the idea to first find patterns in individual signals which can be thought of simple basic driving patterns and subsequently combine these patterns to describe and discover more complex driving behaviors and patterns.

When common driver actions have been identified they are used to label the data set so that similar situations can be classified in new unseen driving data. This is achieved by training a classifier on the labeled data in such a way that the feasibility of implementing the classifier as an online subroutine in the onboard vehicle computer is retained. This led us to transform the data set as a collection of time sequences with corresponding labels from the original data set which would simulate an online classification situation. The choice classifier was different types of artificial neural networks because empirical evidence suggests they are suitable

for this kind of problem. In particular, four different architectures were investigated and compared.

5.1.2 Results

The first part of the thesis where driving patterns and behaviors are discovered yielded a labeling of the time series data and some statistical conclusions could be drawn concerning the driver behavior in different types of intersections. In particular, ten different driving scenarios were identified which described roughly 80 % of the data. From the driving patterns discovered in the steering wheel angle signal, three different types of intersections were found. A verification of this pattern discovery method was performed using GPS data when available. This analysis showed that the overall ground truth agreement for the pattern discovery algorithm was 77 %. Other driving behaviors were analyzed in the intersection scenarios and some statistical tests were performed to draw conclusions about driver actions. The data suggests that there is a statistically significant difference between some behaviors in various intersection situations, in particular, we found that there is often a significant difference in behaviors between roundabouts and left turn intersections.

After obtaining a labeling of the driving data, different architectures of ANNs were trained in a situation simulating online classification of driving scenarios. The ANNs were trained under similar conditions to get a fair comparison of their performance. Since the data set is imbalanced with one class dominating the other, we concluded that accuracy is not the best measure for model performance. To measure the performance of the networks, recall, precision and F1 metrics were also calculated. Each of them evaluate different characteristics of the classification models. For our purpose, we aim for precision over recall since we are looking for low false positive cases for identifying intersections. t-LeNet outperforms the rest of the models with a precision of approximately 95 % for identifying the intersections and it is better at detecting three out of five of the classes. It is followed by the Stacked LSTM model in terms of accuracy and precision. Recall rate is in general low for the four compared models, which lead us to say that there is a high rate of cases where the car fails to detect to be in an intersection.

From the individual label classification, it was observed that, as expected, steady state is the easiest to identify. It is followed by the outlier class, right turn, left turn and finally the roundabout. Each model has different abilities for identifying specific scenarios. One example is the t-LeNet model which outperforms the rest for the outlier identification, while ResNet has better abilities for identifying roundabouts.

5.2 Discussion

In this section we will discuss the impacts of the limitations in this project and the scientific contribution of our work.

5.2.1 Limitations

One of the major limitations in this project is that the labeling and pattern discovery algorithms were developed on a data set consisting of one route driven by one driver. This may potentially have an impact on the results and reliability of some the conclusions drawn. However, we believe that the effect of this limitation should be minimal for a few different reasons, namely, the route contains different road types, such as highway, country side and city driving which should cover most common scenarios. In addition, the data were collected in different driving conditions such as heavy congestion on the route. A simple way to mitigate the limitation of having only one driver is to have another one collecting data along the same route, however, due to time constraints this was not possible during the course of this thesis but may be explored at a later stage.

Another limitation is that we underestimated the amount of time the first part (pattern discovery part) would take which resulted in some less time spent on the last part of the thesis (classification part). We still think the time split between the two different parts is reasonable since the performance on the classification task will be negatively affected by an inadequate labeling. A more severe effect of the allocation of time is that we initially planned to come up with a predictive model for power demand, which then unfortunately had to be omitted. We will, however, give some ideas and suggestions for such a model in the next section.

A minor limitation in the classification part is the computer hardware that was used to train the neural networks. Usually one trains these models most efficiently on GPUs (graphic processing units) for their superior memory bandwidth and ability to parallelize computations. We did not have access to this hardware but we do not think that this has had any effect on the conclusions and results obtained in this thesis.

5.2.2 Contribution

The work done in our thesis has contributed in two main ways. Firstly, we have developed a novel approach to discover and label driving behavior through the process of first finding simple patterns in individual signals which are then combined for discovering more complex driving patterns. Secondly, we have applied time series classification methods to obtain a traffic scenario classification algorithm. This model could be implemented as an online subroutine in the vehicle software[1].

[1]In the very last stages of working on this thesis we were able to implement the classifier as a subroutine in the vehicle software with the help of our Volvo supervisor Chih Feng Lee. The implemented model was the LSTM unit since it allows constraining the number of parameters while not compromising performance.

The first contribution is valuable since there is a large amount of unlabeled data available and more are generated almost daily from the frequent software testing done at the Propulsive Software Department. The methods developed can be useful for discovering, labeling and analyzing more driving data in new routes.

The second contribution is potentially beneficial in several ways. To have an online classifier in the vehicle could help the ECU make intelligent decisions about power distribution, gear shifting and energy recuperation which could give a better driving experience and make the vehicle more efficient. We believe that more onboard prediction and classification software is the way forward when developing modern vehicles. There is potentially a lot to gain from these systems and methods in ways we might not see right now and that makes our work important. As this area is relatively new, we hope that our findings and research may be of help and maybe even inspire others to implement similar algorithms in the future.

5.3 Future work

In this section we will outline some ideas for continuing the work that has been done in this project.

One obvious extension of the work we have done is to test the methods with another driver on another route. We also need to implement the online classification model in a vehicle and test it in an actual driving situation.

Another interesting area for further research is using our classification results for power demand prediction. The assumption is that different driving scenarios have particular associated power demands and thus knowing the current traffic situation can be used to infer the near future driver power request. We have some suggestions for how this can be done which we will briefly discuss here. In particular, during the course of this thesis we have come up we two plausible models for power demand prediction. The first model would be a forecasting model that takes the most recent history of driving inputs, just like our classification model, and tries to predict the power request the coming 5 seconds ahead. The other model would use the knowledge of the current driving situation to predict the near future power demand. The rationale is that each driving scenario has some average power demand associated to it and one could then simply construct a power demand prediction model. This can be achieved by weighting the power demand averages with the probabilities for each traffic scenario as given by the softmax output layer in our classification algorithms. Our idea is to compare the performance of these two approaches and possibly combine them. It would have been interesting to see how that turned out and it is an relevant problem that should be addressed in future work.

Bibliography

[1] MATLAB 2019a, MathWorks, Inc. `findchangepts` - find abrupt changes in signal, 2019. `https://se.mathworks.com/help/signal/ref/findchangepts.html`.

[2] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11. ACM Press, 2003.

[3] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15:107–144, 08 2007.

[4] Eamonn J Keogh and Michael J Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[5] Rui Xu and Donald C Wunsch. Survey of clustering algorithms. 2005.

[6] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[7] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, pages 267–276, 1953.

[8] Yoshiki Tanaka and Kuniaki Uehara. Discover motifs in multi-dimensional time-series using the principal component analysis and the mdl principle. volume 2734, pages 252–265, 07 2003.

[9] Mohammad Hossin and Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 03 2015.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.

[13] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[14] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[15] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *CoRR*, abs/1809.04356, 2018.

[19] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *CoRR*, abs/1611.06455, 2016.

[20] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[23] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How Does Batch Normalization Help Optimization? *arXiv e-prints*, page arXiv:1805.11604, May 2018.

[24] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks. In *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.

[25] WH Kruskal and WA Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, pages 583–621, 1952.

[26] Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

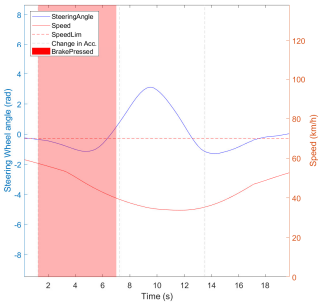
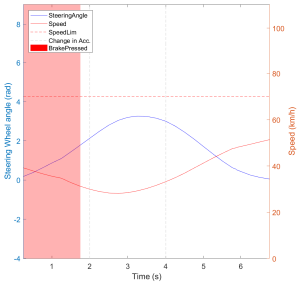
[27] Denis Smirnov and Engelbert Mephu Nguifo. Time series classification with recurrent neural networks.

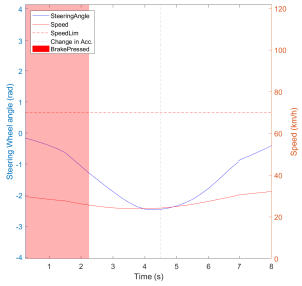
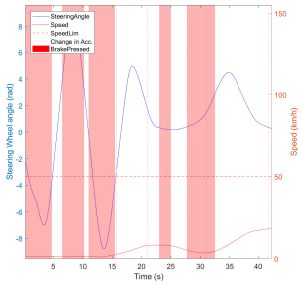
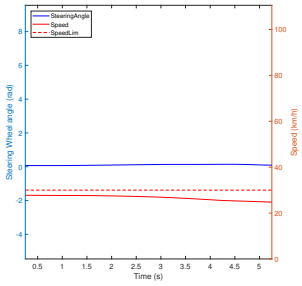
[28] Zhiyong Cui, Ruimin Ke, and Yinhai Wang. Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction. *arXiv preprint arXiv:1801.02143*, 2018.

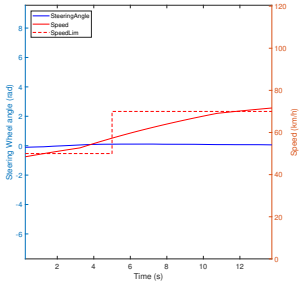
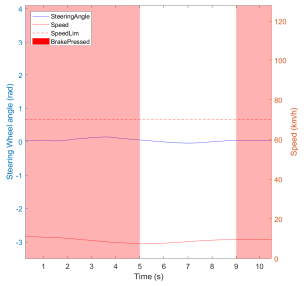
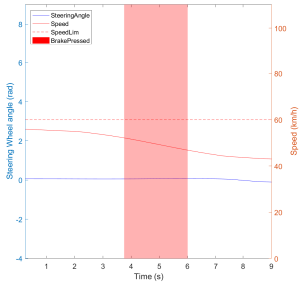
[29] François Chollet et al. Keras. `https://keras.io`, 2015.

A

Appendix 1

Scenario	Description	Information
1)	**Roundabout**	
	The vehicle is in a roundabout intersection. The label is obtained considering only the steering wheel signal.	Repetitions 166 % time spent 5.24% Min. Time 6.5 s Max. Time 417 s Median Time 23 s
2)	**Left turn**	
	The vehicle takes a left turn. The label is obtained by considering only the steering wheel signal.	Repetitions 129 % time spent 2.91 % Min. Time 2.5 s Max. Time 93.25 s Median Time 16.37 s

Scenario	Description	Information
3) **Right turn**		
	The vehicle takes a right turn. The label is obtained by considering only the steering wheel signal.	Repetitions 172 % time spent 3.73% Min. Time 5.25 Max. Time 284 Median Time 15.73
4) **Outlier**		
	Behaviour found when arriving or leaving a parking lot. It is characterized by large turning maneuvers and frequent braking.	Repetitions 30 % time spent 2.64% Min. Time 12.25 Max. Time 323.5 s Median Time 63.9 s
5) **Straight**		
	The vehicle keep a constant speed without braking or turning. This happened commonly when a vehicle drives in a highway.	Repetitions 1503 % time spent 54% Min. Time 5.25 Max. Time 228.75 s Median Time 26.1 s

Scenario	Description	Information
6) **Increase in speed limit**		
	The driver accelerates as they approach an increase in speed limit	Repetitions 203 % time spent 3.14 % Min. Time 1 s Max. Time 35 s Median Time 11.5 s
7) **Congested**		
	The vehicle keeps a constant speed while also braking. This type of behaviour is common when there is traffic congestion and vehicles are waiting in a queue.	Repetitions 124 % time spent 1.71% Min. Time 2 s Max. Time 112 s Median Time 11.5 s
8) **Reducing speed by braking**		
	The vehicle decelerate and then use the braking to reduce the speed even more.	Repetitions 241 % time spent 2.86% Min. Time 1.5 s Max. Time 28.25 s Median Time 8.8 s

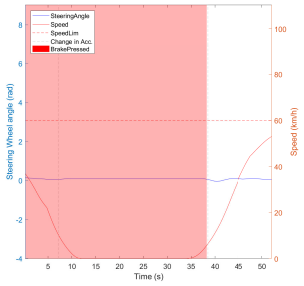
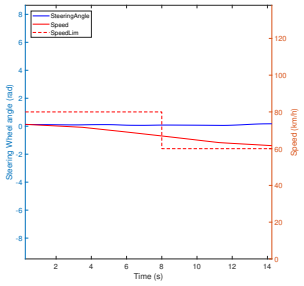
Scenario	Description	Information
9)	**Waiting in traffic**	
	The vehicle brakes and decelerates until stopping. After some seconds, it accelerates. This type of behaviour is commonly observed at traffic lights and queues.	Repetitions 102 % time spent 4.81 % Min. Time 9.75 Max. Time 113.75 s Median Time 34.26 s
10)	**Decrease in speed limit**	
	The vehicle decelerates as a response to a decrease in speed limit	Repetitions 57 % time spent 0.82% Min. Time 2.25 s Max. Time 35 s Median Time 10.5 s

Table A.1: List of scenarios obtained by using the steering angle signal (1-4) and the pattern sequences discovery (5-10). In the last column information about the scenario duration and the number of repetitions in our data set can be observed.