CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Machine Learning Meets Localization

Hypothesis inference for localization of autonomous vehicles

Master's thesis in Computer science and engineering

THEODOR STENHAMMAR
DAVID BEJMER

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Machine Learning Meets Localization

Hypothesis inference for localization of autonomous vehicles

THEODOR STENHAMMAR
DAVID BEJMER

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Machine Learning Meets Localization
Hypothesis inference for localization of autonomous vehicles
THEODOR STENHAMMAR
DAVID BEJMER

Supervisor: Marina Axelson-Fisk, Department of Mathematical Sciences
Supervisor: Axel Beauvisage, Zenseact
Supervisor: Junsheng Fu, Zenseact

Examiner: Marina Axelson-Fisk, Department of Mathematical Sciences

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Machine Learning Meets Localization
Hypothesis inference for localization of autonomous vehicles
THEODOR STENHAMMAR
DAVID BEJMER
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis project was conducted in cooperation with Zenseact for the purpose of creating a solution for determining the lane in which an autonomous vehicle is driving. Solving this is part of the larger problem of localization and state estimation of autonomous vehicles and is referred to as Lane-Level Localization (LLL).

The problem is connected to the area of Early Time Series Classification, which is the field of applying supervised learning and time series classification techniques for classifying time series accurately with as few observations as possible.

The problem of LLL may be solved by applying what is known as a multi-hypothesis technique. This is a technique that estimates some state by tracking several different possibilities (hypotheses) for the state and using some model for inferring the most likely scenario.

It is found that using an architecture that allows for the possibility of rejecting output depending on the certitude with which a classification can be made can be adapted to solving the problem of LLL in autonomous vehicles. In the current scenario, the model produced an accuracy of 99,5% while only rejecting to classify in 1% of the sequences.

Keywords: Engineering, thesis, Zenseact, machine learning, localization, autonomous driving, data, time series, early, early classification, lane-level localization.

Acknowledgements

We would like to thank Zenseact and our supervisors Axel Beauvisage, Junsheng Fu, and Marina Axelson-Fisk for their help and guidance in the project and the writing of the thesis.

We would also like to thank our peer-reviewers Jakob Hendén and Alfred Arvidsson.

David Bejmer & Theodor Stenhammar, Gothenburg, May 2022

Glossary

Below is a list of the terms and concepts that are referenced in this thesis.

AD	Autonomous Driving
ADAS	Advanced Driver-Assistance Systems
ASF	Achievement Scalarization Function
catch22	Feature set that is derived from the *hctsa* feature set for time series classification.
CIF	Canonical Interval Forest. Ensemble method for categorization of time-series data. Uses the *catch22* feature set.
Ego-vehicle	The physical vehicle from which sensor measurements are gathered, as opposed to a hypothetical vehicle pose.
ETSC	Early time-series classification. The methods relating to classification of partial time-series with accuracy and earliness.
GB	Gradient Boosting. A tree ensemble machine learning algorithm.
GNSS	Global Navigation Satelite System.
hctsa	Highly Comparative Time-Series Analysis. A feature set for time series analysis.
HD-map	A map with precise information about lane geometry, lane marker types, down to centimeter level precision
IMU	Intertial Measurement Unit.
INS	Interial Navigation System.
KF	Kalman Filter. An optimal state estimator. Exists in several variations with different properties.
LSTM	Long Short-Term Memory cell. A type of recurrent neural network cell that is designed to capture information over longer timescales.
MQ	Model qualities. A measurement of similarity between the HD-map and perceived lane markers
MTS	Multivariate time-series. A time-series in which each data point is multi-dimensional.
NSGA2	Non-Dominated Sorting Genetic Algorithm 2. A genetic algorithm for producing non-dominated solutions in multi-objective optimization.
SVM	Support Vector Machine.

Contents

List of Figures **xiii**

List of Tables **xvii**

1 Introduction **1**
1.1 Background . . . 1
1.2 Aim . . . 3

2 Literature Review **5**
2.1 Localization in autonomous driving . . . 5
2.2 Classification of time series data . . . 6
2.2.1 Classification approaches . . . 6
2.2.2 Early time-series classification . . . 7
2.2.2.1 *Early classification of time series using multi-objective optimization techniques* by Mori et al. (2019) . . . 8
2.2.2.2 *TEASER: early and accurate time series classification* by Schäfer and Leser (2020) . . . 10
2.3 Multi-objective Optimization . . . 10

3 Data set **13**
3.1 Data description . . . 13
3.1.1 Main feature set . . . 14
3.1.2 Supplementary feature set . . . 17
3.2 Data preprocessing and analysis . . . 18
3.2.1 Outliers and Scaling . . . 19
3.2.2 Missing data . . . 19
3.2.3 Data visualisations . . . 20
3.3 Data split . . . 23
3.4 Annotation . . . 24

4 Method **29**
4.1 Experimental setup . . . 29
4.2 Definition of a multivariate time series . . . 29
4.3 Solution design . . . 30
4.4 Evaluation . . . 31

5 Experiments and results **33**
5.1 Probabilistic classifier . . . 33
5.1.1 Momentary measurement model qualities . . . 34
5.1.2 Averages . . . 36
5.1.3 Constructed feature set . . . 38
5.1.4 State-of-the-art methods and candidate selection . . . 42
5.2 Trigger function . . . 43
5.2.1 Multi objective optimization . . . 43
5.2.1.1 Objectives . . . 43
5.2.1.2 Optimization algorithm . . . 44
5.2.1.3 Trigger functions . . . 45
5.2.1.4 Trigger function parameter optimization . . . 46
5.2.1.4.1 Tr^1 . . . 46
5.2.1.4.2 Tr^2 . . . 47
5.2.1.4.3 Tr^3 . . . 49
5.2.2 Testing of trigger functions . . . 50
5.2.2.1 Highlighted cases . . . 51
5.2.2.1.1 Incorrect classifications . . . 51
5.2.2.1.2 Late prediction . . . 52
5.2.2.1.3 No prediction . . . 54

6 Discussion and Conclusion **57**
6.1 Results discussion . . . 57
6.1.1 Data . . . 57
6.1.2 Probabilistic classifier . . . 58
6.1.3 Trigger function . . . 58
6.1.4 Hypothesis-inference . . . 59
6.2 Limitations and scope . . . 60
6.3 Conclusion . . . 60

Bibliography **61**

List of Figures

1.1 The robotic paradigm divides the continuous control loop into sensing the environment, planning for the future, and taking an action. 1

1.2 Example of multiple hypotheses. The green vehicle is the reference state of the vehicle and the beige vehicles are hypothetical states. . . 2

2.1 Selection process using reject classifier as described in Mori et al. (2019). $TS|_t$ is a time-series prefix up to time t. p_k^t is the k:th highest probability (i.e. the k:th best hypothesis according to the probabilistic time-series classifier. A trigger output of 1 signifies a selection of the hypothesis corresponding to the highest probability p_1^t). 9

2.2 A Pareto-front, or non-dominated set of solutions, shown by the red line. Every square represents some selection of parameters and the performance is evaluated over two dimensions, represented by the coordinate axes. For every selection of parameters that is part of the Pareto-front, no objective can be improved without a decrease in the other objective. 11

3.1 Initialization of hypotheses based on GNSS. The blue and yellow ellipses show the uncertainty in the GNSS signal at the time of initialization. One hypothesis is initialized for each lane and they are numbered according to the bottom of the image. 14

3.2 Example of perception overlaid on HD-map. Red color is the perceived lane markers, placed with hypothesis 1 as reference point. The green colored lines are the exact same as the red lines, except from the perspective of hypothesis 0. Note the mismatch of the lane types when the perceived lane markers are overlaid over hypothesis 1. . . . 16

3.3 Example of perception overlaid on HD-map. Red color is the perceived lane markers, placed with hypothesis 1 as reference point. The green colored lines are the exact same as the red lines, except from the perspective of hypothesis 0. Note that the geometry of the perceived lane markers differ significantly from the HD-map. 17

3.4 Boxplot showing a box of Q1-Q3, the median value as a green horizontal line and whiskers reaching at the most 1.5 times the length of the box. Outliers are presented as circles. 18

3.5 Momentary MQs for samples coming from correct and false hypotheses respectively. The distributions are normalized to show density, i.e. the total area of each of the two distributions sums to 1 (y axis). On the x axis is the MQ-value. The distributions are presented by a histogram of 50 bins over [-10, 5] and the data is split into two colors by label. 20
3.6 Pairwise correlation matrix of the features and the target label using Pearson correlation coefficient. A more blue color means a stronger positive correlation while a more red color means a stronger negative correlation. A value closer to 0 or a lighter color closer to white means no correlation. Any absolute correlation value below 0.5 is considered more or less weak. 21
3.7 Averaged version of the feature set, 50 bins over [-10, 5]. Color by label. The average is taken as the mean value for each feature over whole sequences. 22
3.8 Low-Pass version of the feature set, 50 bins over [-10, 5]. Color by label. 22
3.9 Map showing the three different data sets in the colors red, blue and green. They have been split into different sets based on the angle from the reference point marked with yellow and their mean position. 24
3.10 Example annotation. The white shapes show the pose of the possible hypotheses and the green shape show the pose of the reference vehicle. The arrows signify lane center line and direction of movement. 26
3.11 Two annotation scenarios. In the left scenario, $\Delta y_0 << \Delta y_1$ and hypothesis 0 is annotated as the correct hypothesis. In the right scenario, y_0 and y_1 are similar, and no annotation is made for that frame. 26
3.12 Example of lane merge leading to a convergence of hypotheses and ambiguity in annotation. Reference vehicle is drawn behind the hypotheses for clarity, as longitudinal offset is not considered for annotation purposes. 27

4.1 An overview of the full solution architecture, based on the process in Mori et al. (2019). $MTS|_t^n$ is the n:th time series prefix, up to time t. p_k is the predicted true-probability as given by the probabilistic classifier for hypothesis k. A trigger output of 1 signifies an output of the hypothesis corresponding to the highest probability in P. . . . 30

5.1 Solution architecture with the probabilistic classifier highlighted. . . . 33
5.2 Line plot showing the accuracy for different classifiers and amount of training data. The models in the left figure are trained and tested on momentary MQs with missing values replaced with the constant 5. The models in the right figure are trained and tested on momentary MQs with missing values replaced with the mean value for the given feature in the training set. 35
5.3 Bar plot showing feature importances extracted from the trained Gradient Boosting classifier. All values sum up to 1. 36

5.4 Line plot showing the 5-fold cross validation accuracy for different time steps and differently trained GB classifiers, averaged over ten runs. The legend describes what each GB classifier has been trained on. Numbers in parentheses describe the size of the moving window, over which it averages. 37

5.5 Correlation matrix of the feature set as described. A high value means that the features have a positive correlation while a low value means that they have a negative correlation. 40

5.6 Results by training on constructed feature set using a Gradient Boosting classifier with default parameters. Evaluation done on time-series prefix lengths between [40, 1201]. 41

5.7 Feature importances for the using the Gradient Boosting classifier. All importance-values sum to 1. 41

5.8 Line plot showing the 5-fold cross validation accuracy for CIF-classifiers trained on six different time series sizes compared to a GB classifiers trained on a moving average of 200 and predicting on current average as well as a GB classifier trained and tested on a constructed feature set. The graph shows the accuracy for a given model at a given time step. 42

5.9 Solution architecture highlighting the trigger function. 43

5.10 Recap of the geographic location of the different data sets. As can be seen, the red data set is largest, with the green and blue data sets being around a third to a half as large as the red. 45

5.11 The figure shows results from the optimizations on the three different optimization data sets when using Tr^1. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost. . . 47

5.12 The figure shows results from the optimizations on the three different optimization data sets when using Tr^2. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost. . . 48

5.13 The figure shows results from the optimizations on the three different optimization data sets when using Tr^3. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost. . . 49
5.14 Scatter plot showing the performance of the three trigger functions with their optimized parameters on the testing data. The x-axis describes the fraction of correct predictions, the y-axis shows the average waiting time to prediction, in terms of the fraction of 30 seconds, and the availability of a prediction is presented in the legend together with the tested trigger function. 51
5.15 Model Qualities for the correct hypothesis to the left and the incorrect but selected hypothesis to the right. The y-axes shows the MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected. 52
5.16 Averaged Model Qualities for the correct hypothesis to the left and the incorrect but selected hypothesis to the right. The y-axes shows the averaged MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected. 52
5.17 Model Qualities for the correct (and highest probable) hypothesis to the left and the second most probable hypothesis to the right. The y-axes shows the MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected. 53
5.18 Averaged Model Qualities for the correct (and highest probable) hypothesis to the left and the second most probable hypothesis to the right. The y-axes shows the averaged MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected. 53
5.19 The two highest probabilities from the probabilistic classifier, p' and p" respectively, for the time steps before prediction. The black vertical line represents the time step at which a prediction was triggered. . 54
5.20 The two highest probabilities from the probabilistic classifier, p' and p" respectively, for a sequence in which the trigger function never triggered a prediction. 54
5.21 The two highest probabilities from the probabilistic classifier, p' and p" respectively, for a sequence in which the trigger function never triggered a prediction. 55

List of Tables

3.1 Table of metrics for the primary feature set. The minimum, mean, standard deviation, maximum and proportion of present values are shown for each feature in the main feature set (Model Qualities). . . . 18
3.2 Table showing the mean and standard deviation of the data set when split on the target label, meaning if a given row is produced by a true or false hypothesis. 21
3.3 Table showing the number of sequences included in each data set as well as the mean and standard deviation for each feature. 24

5.1 Table presenting how the different data sets are used in each of the three experiments. Train means the training of the probabilistic classifier, Optimize is the optimization of the trigger function parameters and Test is the independent data set on which the whole solution is applied. 45
5.2 The learned parameters by optimizing on three different data sets for Tr^1 47
5.3 Table showing the learned gamma parameters from the optimization on the three different data sets for Tr^2. 49
5.4 Table showing the learned gamma parameters from the optimization on the three different data sets for Tr^3. 50

1

Introduction

> *"... a fundamental aspect of a fully autonomous vehicle is its ability to properly perceive its environment and localize itself on the road"*

This was written by Laconte et al. (2022) with regards to *Advanced Driver-Assistance Systems* (ADAS). A localization system needs to be precise in its perception of the vehicle state as this information will be used as foundation for control decisions in *Autonomous Driving* (AD). For performing a maneuver such as overtaking another vehicle, there can be no uncertainty about the position of the vehicle.

1.1 Background

An autonomous vehicle is a robot and follows the *Sense-Plan-Act*[1] (SPA) paradigm. *Sense*, in this context, means the robot's ability to gather and interpret information from the surrounding environment. This depends on the hardware capabilities that the robot has as well as how software is used to derive key information about the environment and the state of the robot. Localization of a robot means finding the state of a robot in relation to some environment, and is a key part of "Sense".

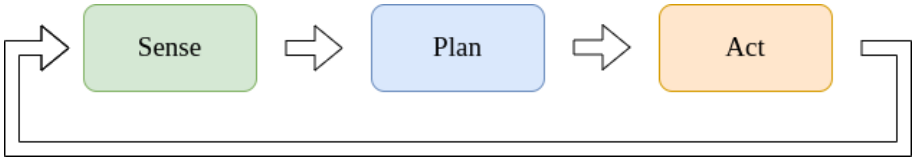


Figure 1.1: The robotic paradigm divides the continuous control loop into sensing the environment, planning for the future, and taking an action.

GNSS devices are insufficiently precise for localization of an autonomous vehicle [2]. Some other method is therefore necessary in order to localize a vehicle.

[1] This was also the inspiration for the company name; *Zenseact*. For information on SPA see Lauxmann (2015).

[2] For perspective, GNSS-devices are commonly cited to have 95% accuracy in localizing an object within 3 meters. For further information on GNSS usage in autonomous driving, see Joubert et al. (2020).

Laconte et al. (2022) distills the task of localization in autonomous driving into three smaller components; *Road Level Localization* (RLL), *Ego-Lane Level Localization* (ELL), and *Lane-Level Localization* (LLL). RLL means identifying the road on which the vehicle travels and ELL considers the position of the vehicle within the lane. LLL concerns the task of identifying in which lane of the road the vehicle is traveling.

This thesis project is focused on solving part of the Lane-Level Localization problem for autonomous driving. The project is performed in cooperation with Zenseact, a software company that specializes in ADAS and AD solutions and provides such services for Volvo. Zenseact works with the entire software solution for AD and ADAS, the full Sense-Plan-Act loop of autonomous driving. Localization of the car is part of the vehicle state estimation and is therefore a baseline from which decisions can be taken.

For this project, we employ a version of *Multiple-Hypothesis Tracking* for lane-level localization. This method was first proposed by Reid (1979) and uses the idea of creating multiple possible hypotheses for the states of each target, and removing the hypotheses that are unlikely to be true. The multiple-hypothesis tracking and related techniques have been used for localization purposes in many cases, for instance by Jensfelt and Kristensen (2001) and by Jensfelt and Kristensen (2001).

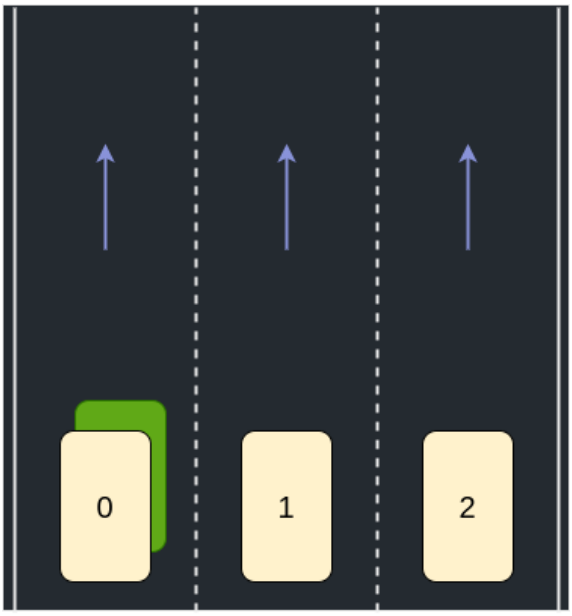


Figure 1.2: Example of multiple hypotheses. The green vehicle is the reference state of the vehicle and the beige vehicles are hypothetical states.

The use of multiple-hypothesis tracking solves the LLL problem by dividing it into two sub-problems: the tracking of multiple hypotheses for the state of the vehicle, and inferring the correct state out of the set of possible hypotheses. The tracking of the hypotheses is accomplished using a *state-estimator*, for example a Kalman filter. An example of multiple-hypothesis tracking for LLL is seen in figure 1.2.

Creating an inference model for selection between these hypotheses is the subject of this thesis project.

1.2 Aim

To make safer decisions a vehicle must know where it is in the road, and therefore, robust and quick initialization of the localization is of importance. The aim of this thesis project is to create an inference model using supervised machine learning techniques for the task of online lane-level vehicle localization by classifying a number of state hypotheses. The aim can be further specified as:

1. Literature review: Investigate state-of-the-art methods for machine learning and localization of autonomous vehicles.

2. Domain analysis: Perform an analysis of the data and a specification of the problem to be solved.

3. Problem specification: Based on literature study and domain analysis, propose and examine approaches to hypothesis inference.

2

Literature Review

Section 2.1 includes further information on techniques for lane-level localization and state estimators. Section 2.2 includes information on techniques for time series classification and the field of Multi-Objective Optimization is presented in section 2.3.

2.1 Localization in autonomous driving

Laconte et al. (2022) divide approaches to Lane-Level Localization into two paradigms; *Landmark Approaches* and *Map-Aided Approaches*. In Landmark approaches, visual features like lane markings and traffic signs are extracted from images and then used to assess the number of lanes and the lane in which the vehicle is driving. Map-aided approaches use environment perception based on sensor data to match the location of the vehicle with a High-Definition map. An *HD-map* is a map that contains detailed information about lane geometries, lane marker types, and other information about a road network.

Map-Aided approaches use a map-matching algorithm to determine the lane in which the vehicle is driving. The type of map that is best suited for this task is a *mesoscale* map. This is a map that is between a microscale map and a macroscale map in precision, providing a suitable trade-off between precision and density. Macroscale maps have metric precision while microscale maps have centimeter accuracy.

Many different approaches to map-matching exist. Common approaches include using a *Particle Filter* (PF) as done by Jo et al. (2017). This approach matches the particles to a segment of the map. The particles are clustered by matched lane segment and an hypothesis for the vehicle pose, position and orientation, is estimated based on each cluster. A similar method is used by Li et al. (2018).

Another approach to vehicle state estimation is using a Kalman Filter. Whereas Particle Filters use Monte Carlo [1] simulation to estimate the state of the vehicle, a Kalman filter uses the assumption of linear state transitions and Gaussian-distributed noise. The estimation of the vehicle state is achieved by interpolating

[1] Monte Carlo methods are methods that use random sampling to estimate a result or a distribution.

between measurements of the hidden vehicle state and the expected position based on the model of the linear system, using the covariances of the measurement and the model.

The standard Kalman filter does not rely on simulations and is therefore faster. There are variations to the Kalman filter to handle non-linear state transitions. Examples of this are the *Extended Kalman Filter* (EKF), the *Unscented Kalman Filter* (UKF), and the *Cubature Kalman Filter* (CKF).

For solving the Lane-Level Localization problem using Kalman filters, a filter is initialized for each possible location given some initialization strategy, signifying the different possible states of the vehicle. The solution to the problem consists of inferring the correct hypothesis out of the set of possible hypotheses.

2.2 Classification of time series data

General approaches for classification of time series, as well as relevant considerations are presented in section 2.2.1. The field of Early Time Series Classification is introduced in section 2.2.2.

2.2.1 Classification approaches

We refer to tabular data as data that can be structured as rows and columns in a table where each row represents a single item and the columns represent the various measurements that make up that item. This is a common way of structuring and representing data as it is versatile and can suit many different purposes. This structure makes no assumption on the relation between different items of data.

Sequential data, on the other hand, can also be represented as a table using rows and columns, but there exists a sequential relation between the items. This can be observations over time, letters in a word, or any other sequence of items.

Methods for classification of time series data often differs from methods for classifying tabular data. The reason for this is that the key information is in the sequential relation in the data. The goal is to use a classification approach that captures this information (Bagnall et al., 2017).

Like for any machine learning task, the type of data will determine which methods are appropriate. For instance, sequential data can be discrete or continuous, as well as univariate or multivariate, and this will need to be considered when choosing a machine learning approach (Ruiz et al., 2021).

Time series data is a special case of sequential data, where the sequence is ordered by time. Any methods that are used for classification of sequential data may therefore be of relevance for the purpose of time-series classification. *Long Short-Term*

Memory cells (LSTM)[2] have been utilized in the field of natural language processing in many cases such as Wang and Jiang (2015) and Yao and Guan (2018) but the technique has also been applied specifically to time series classification in Huang and Li (2021).

Historically, a common benchmark for time series classification has been the *DTW-1NN* classifier. Dynamical Time Warping (DTW) is a similarity measurement between time series that can be paired with a 1-Nearest Neighbor classifier (1-NN) for classification of time series Bagnall et al. (2017). The DTW similarity measurement has also been adapted for measuring the similarity between multivariate time series (Ruiz et al., 2021).

Time series classification can be done on raw data, or it can be done using some feature extraction or transformation method. Examples of features that can be extracted from time-series data are the mean of values, the variance, or the mode of the data.

An example of using a feature extraction (or feature transformation) for time series classification is ROCKET by Dempster et al. (2019). This method transforms time series features by extracting features using random convolutional kernels, which is convolutional kernels with random weights, as opposed to learned weights.

Time Series Forest (TSF) by Faouzi and Janati (2020) is a model for time-series classification by use of feature extraction. TSF divides the time-series into windows and extracts the *mean*, the *standard deviation* and the *slope* of the data within that window. Classification is then performed using a tree ensemble over the extracted features.

Another example of feature based time-series classification is the *Canonical Interval forest* (CIF) as proposed by Middlehurst et al. (2020). The Canonical Interval Forest uses a time-series feature set called *catch22* first developed by Lubba et al. (2019). The catch22 feature set is based a filtered version of the *Highly Comparable Time-Series Analysis* (hctsa) feature set by Fulcher and Jones (2017) which originally comprises over 7700 time-series features. The catch22 set is a selection of 22 features from the hctsa feature set with the purpose of selecting as few features as possible while sacrificing as little classification power as possible, in the interest of reducing complexity and training time.

2.2.2 Early time-series classification

Santos and Kern (2017) defines *Early time-series classification* (ETSC) as:

> *"... the problem of trying to come to a classification decision with as little observations of a time series as possible, while sacrificing classification accuracy as little as possible"*

[2] A form of Recurrent Neural Network (RNN)

The idea is to use ETSC-techniques in situations where earliness is key. Examples of application areas include detecting gas-leakages, which was done in an early application of ETSC-techniques by Hatami and Chira (2013) and classification of drug response in Multiple Sclerosis patients in Ghalwash et al. (2012).

The goal is to make a prediction as soon as it can be made with certitude, with the idea that the more observations are made, the more certain a prediction is. The problem becomes finding methods for managing this trade-off between accuracy and earliness. ETSC can also be achieved by a variety of different ways. Santos and Kern (2017) provide a review of some approaches to ETSC.

Another consideration is that ETSC often requires a different approach to classification than offline time series classification. The reason for this is that in ETSC the full time series is not available. Some methods for time series classification, such as the Multi-Layer Perceptron for classification of time series as proposed by Wang et al. (2016), are dependent on the length of time series. A method that is independent of the length of the time series can be used any time new data is collected, but a method that is dependent on the length can only be used for the specific time series length that it is constructed for. The application of such length-dependent methods for ETSC requires defining a set of lengths at which time series are evaluated and constructing a set of classifiers, one for each length.

An approach that been used by Schäfer and Leser (2020) and Mori et al. (2019) consists of a two-tiered structure. This is based on dividing a time-series into segments of different lengths and using a probabilistic classifier trained to output class probabilities for different lengths of time series segments. This is then combined with a reject-classifier that chooses to output a prediction or wait for more data depending on the certitude of the prediction.

2.2.2.1 *Early classification of time series using multi-objective optimization techniques* by Mori et al. (2019)

Mori et al. (2019) propose a novel approach to multi-objective optimization for early time series classification which involves optimizing accuracy and earliness by training the parameters of a trigger function that makes the decision of predicting or waiting for more data. The approach by Mori et al. (2019) is shown in figure 2.1.

The purpose of the trigger function is to be responsible for deciding if the output from the probabilistic classifier is reliable enough to make a prediction (output of 1) or wait for more data (output of 0). If it deems the prediction to be reliable enough, then the hypothesis with highest posterior probability is chosen. Mori et al. (2019) proposes using a linear trigger function. The linear trigger function has been shown to be competitive against more complex rules and state-of-the-art methods.

$$Tr_{\boldsymbol{\gamma}}(\mathbf{p}_t, t) = \begin{cases} 0, & \text{if } \gamma_1 p'_t + \gamma_2(p'_t - p''_t) + \gamma_3 \frac{t}{L} \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.1}$$

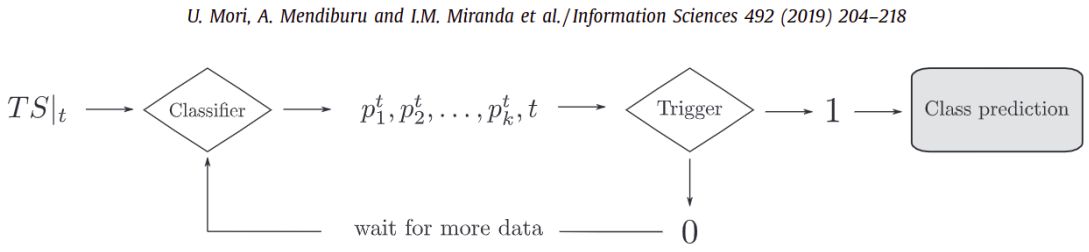


Figure 2.1: Selection process using reject classifier as described in Mori et al. (2019). $TS|_t$ is a time-series prefix up to time t. p_k^t is the k:th highest probability (i.e. the k:th best hypothesis according to the probabilistic time-series classifier. A trigger output of 1 signifies a selection of the hypothesis corresponding to the highest probability p_1^t).

$\mathbf{p}_t$ is the probability vector obtained at the current time step t and with length equal to the amount of active hypotheses. p_t' and p_t'' are the largest and second largest probability respectively in this vector, and $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \gamma_3)$ is the vector of parameters that needs to be chosen, where $\gamma_i \in [-1, 1]$ for all i. L is the length of the time series, so that $\frac{t}{L}$ is the fraction of the time series that has been observed at time t.

The algorithm used to perform the multi-objective optimization is the non dominated sorting genetic algorithm NSGA2, which is a genetic algorithm for multi-objective optimization first proposed by Deb et al. (2002).

The accuracy cost is defined as the classification error or percentage of wrongly classified sequences:

$$C_{ac}(X, Tr_\gamma) = \frac{1}{|X|} \sum_{x \in X} \iota(\hat{y_x} \neq y_x) \tag{2.2}$$

where ι as above is a Boolean function which takes a value of 1 if the condition is true and 0 otherwise. X is the set of sequences (time series) on which to evaluate. Tr_γ is the trigger function as described in Equation 2.1, parameterized by γ.

The earliness cost is defined as the average length of the time series portion that has been observed at the time of prediction:

$$C_{ea}(X, Tr_\gamma) = \frac{1}{|X|} \sum_{x \in X} \frac{t_x^*}{L_x} \tag{2.3}$$

Where X and Tr_γ is as described above, t_x^* is the time step for which an hypothesis selection is available for x using Tr_γ, and L_x is the length of x.

2.2.2.2 *TEASER: early and accurate time series classification* by Schäfer and Leser (2020)

The solution proposed by Schäfer and Leser (2020) is similar to the solution proposed by Mori et al. (2019).

TEASER employs a two-tiered architecture where the first tier produces class probabilities, and the second tier decides whether a selection can be made with. A set of first tier- and second tier-classifiers are trained for specific lengths of data.

The earliness and accuracy definitions are similar to the ones found in Mori et al. (2019). The difference is that they are not phrased as "costs" for the purpose of multi-objective optimization. The accuracy is therefore phrased as the proportion of correct predictions rather than the proportion of incorrect predictions. The earliness is phrased as the average proportion of remaining data, rather than the average proportion of observed data.

These measurements are also supplemented by the harmonic mean:

$$HM = \frac{2(1 - earliness)accuracy}{(1 - earliness) + accuracy} \tag{2.4}$$

As the time series are of variable lengths, it is noted by the authors that the time series that are trained on longer segments of time series will have less data available. For this reason, they choose to use a single class classifier, trained on each class.

2.3 Multi-objective Optimization

There are a few advantages to optimize over the two objectives separately instead of merging them together into one objective. Firstly, with a single objective, there would be a need to weight the importance of the availability and accuracy in advance, which according to Mori et al. (2019) may be difficult in some situations. Moreover, no consideration needs to be made regarding the scaling of the measures, which wouldn't be the case if they were to be optimized together. Finally, the results from a multi-objective execution give a clearer picture of the availability-accuracy trade-off without having to run the algorithm several times as would be the case in the single-objective scenario.[3]

The results are given in the form of a Pareto-front or a set of non-dominated solutions, i.e. for any solution, no one objective can be improved without a detriment to the other objective.

[3]For further information on multi-objective optimization, see Deb and Deb (2014).

A definition of Pareto-optimality is presented by Marler and Arora (2004) as follows:

> *"A point, $\boldsymbol{x}^* \in \boldsymbol{X}$, is Pareto optimal iff there does not exist another point, $\boldsymbol{x} \in \boldsymbol{X}$, such that $\boldsymbol{F}(\boldsymbol{x}) \leq \boldsymbol{F}(\boldsymbol{x}^*)$, and $F_i(\boldsymbol{x}) < F_i(\boldsymbol{x}^*)$ for at least one function"*

A Pareto-front is shown in figure 2.2.

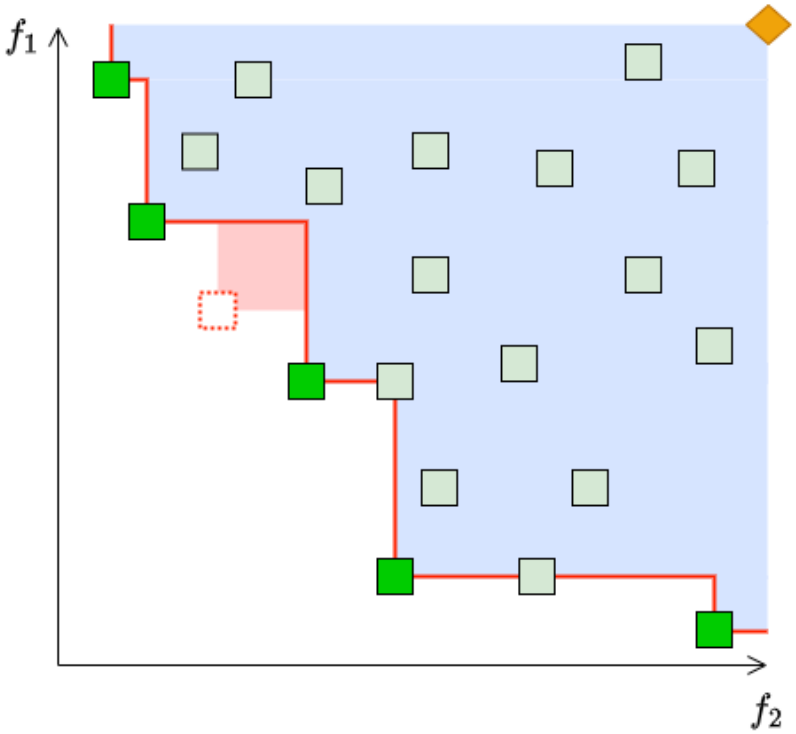


Figure 2.2: A Pareto-front, or non-dominated set of solutions, shown by the red line. Every square represents some selection of parameters and the performance is evaluated over two dimensions, represented by the coordinate axes. For every selection of parameters that is part of the Pareto-front, no objective can be improved without a decrease in the other objective.

The dark green squares are the solutions that are part of the Pareto front. Notice that for every light square, there is some other green square that is at least just as good in both objectives.

The hypervolume is a performance-indicator used in multi-objective problems and coupled to the Pareto-front. For two objectives, the hypervolume is the surface area given by the non-dominated solutions in the Pareto-front and a reference point. This is represented by the light blue area in figure 2.2 and is calculated with respect to some reference point, represented here as the orange diamond.

The hypervolume can be used to visualize the progress made by an algorithm and

give a hint towards if the algorithm has converged, and therefore if the optimization can be stopped.

Consider finding some new solution at the location of the dotted red square. Such a solution would not be dominated by any solution in the Pareto front and would therefore be added to the optimal set. The increase to the hypervolume measurement would be equal to the shaded area above and to the right of the new solution. An increase to the hypervolume means a new non-dominated solution has been found.

3

Data set

The data used in this project is a data set provided to us by Zenseact and it was collected in the months between September 2020 and May of 2021. It has been collected during several different trips on highways in the vicinity of the city of Gothenburg.

Section 3.1 provides a description of the data domain. Section 3.2 contains an exploration of the data and section 3.2.3 provides an analysis of the data for the purposes of classification. In section 3.3, the split of the data into training and validation sets is described and motivated.

3.1 Data description

The data set consists of time series of data describing aspects of a driving scenario. The time series are segmented into sequences of 30 seconds where each sequence starts with an initialization of hypotheses. This involves initializing one Kalman Filter for each lane in the map. For each filter, longitudinal position is determined by the GNSS position while the lateral (in-lane) position is determined by the left and right lane markers. The vehicle heading is determined by direction of the tangent along a projected lane segment.

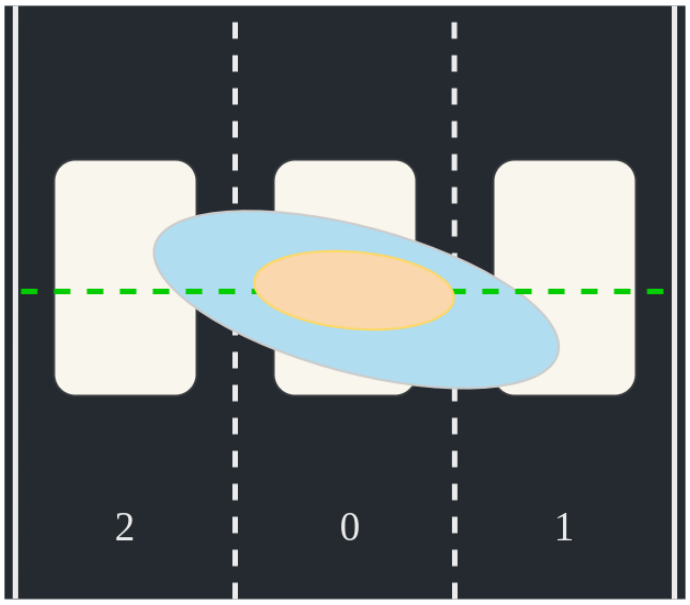


Figure 3.1: Initialization of hypotheses based on GNSS. The blue and yellow ellipses show the uncertainty in the GNSS signal at the time of initialization. One hypothesis is initialized for each lane and they are numbered according to the bottom of the image.

There are 553 sequences, totaling around four and a half hours of driving. Each sequence contains up to 6 hypotheses of varying length. Not all hypotheses are present for the full 30 seconds as they may have been deactivated early. An early deactivation of an hypothesis can be due to the hypothesis running out of the map, which happens when the road parts and there is no map available for the road the hypothesis is on.

The data can be split into a main feature set and a supplementary feature set as described in the following sections.

3.1.1 Main feature set

The features in the main feature set are similarity measurements represented by so called *Model Qualities* (MQs), which describe how well the models used to process vehicle sensor measurements correspond to the actual environment, as described by the HD-map and relative to the position of the hypothesis.The features are part of the same numerical domain, $(-\infty, 0]$, where a lower value means that the sensor measurement for a given feature matches the HD-map worse, from the perspective of a given hypothesis. Any missing MQ is represented by the arbitrarily chosen constant 5.

Following are the six different features which are all collected from the front facing camera:

- Left primary lane marker geometry

- Right primary lane marker geometry
- Left primary lane marker type
- Right primary lane marker type
- Left secondary lane marker geometry
- Right secondary lane marker geometry

Lane marker geometry represents how well the geometry of the lane markers perceived by the car sensors follow the expected geometry as displayed by the HD-map. "Primary" in this case means the lane markers of the lane the hypothesis is located in, and "secondary" means the lane markers of the adjacent lanes. *Lane marker type* shows how well the perceived type of the primary lane markers conform to the lane marker types in the HD-map. Lane marker types could be for example solid, dashed, etc.

This set comes in two versions, filtered and unfiltered MQs. The filtered MQs are MQs that have been processed with a low-pass filter which smooths the MQs over time. They share a domain with the regular MQs.

There may be up to six running hypotheses in parallel, each with their own set of MQs as mentioned above. The Model Qualities are delivered at a frequency of 40Hz.

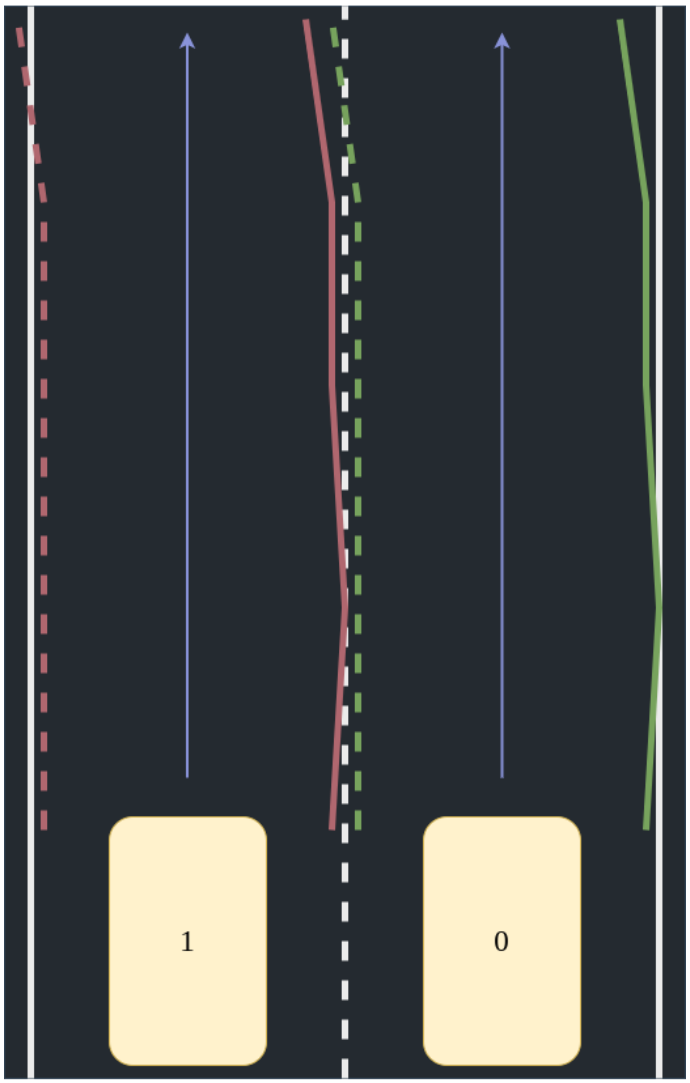


Figure 3.2: Example of perception overlaid on HD-map. Red color is the perceived lane markers, placed with hypothesis 1 as reference point. The green colored lines are the exact same as the red lines, except from the perspective of hypothesis 0. Note the mismatch of the lane types when the perceived lane markers are overlaid over hypothesis 1.

Figure 3.2 shows the perceived lane markers and their type, overlaid on the HD-map. The Model Quality geometries are based on the difference between the perceived lane geometries compared to the HD-map. It can be seen clearly how the type MQs will have a good value for hypothesis 0 and a poor value for hypothesis 1. In this case, it can be inferred from the lane marker types that the ego-vehicle occupies the right lane.

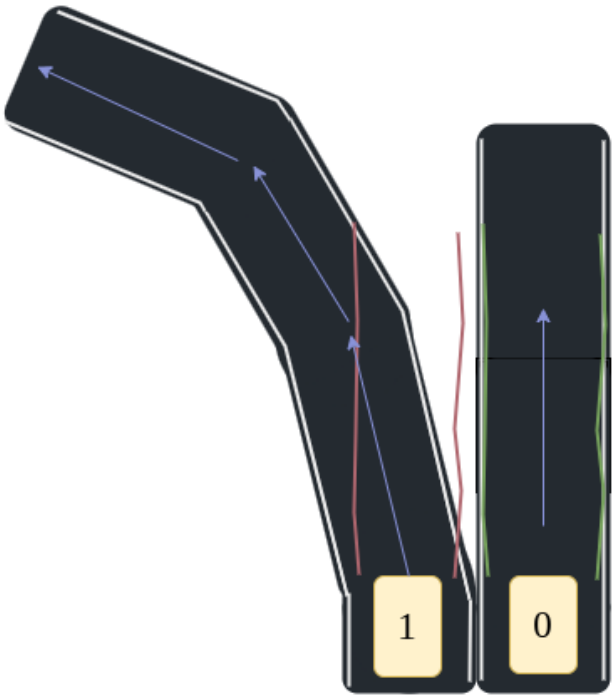


Figure 3.3: Example of perception overlaid on HD-map. Red color is the perceived lane markers, placed with hypothesis 1 as reference point. The green colored lines are the exact same as the red lines, except from the perspective of hypothesis 0. Note that the geometry of the perceived lane markers differ significantly from the HD-map.

Figure 3.3 shows a case where the lane marker types are the same, but the geometry differs significantly due to that hypothesis 1 is on an off-ramp on a highway. Given this information, it can be inferred that the ego-vehicle occupies the right lane.

3.1.2 Supplementary feature set

Beyond the main features, the data set also contains several pieces of supplementary information shown below:

- Coordinate data
- Lane configuration
- Topological information

The coordinate data contains the latitude and longitude for each different hypothesis as given by the Kalman filters, based on GNSS and INS. Among the supplementary features, this is the only feature which is specific for each hypothesis.

Lane configuration is supplementary ground truth information about which lane the car occupies. It has an update frequency of 1Hz, as opposed to 40Hz which the rest

of the data set has. This feature is discrete in the integers from 0 to 6.

Topological information is a categorical feature explaining the surrounding road and includes lane splits or merges and whether there are on- or off-ramps in the area. This information is included as it affects the initialization of the Kalman filters. They will not initialize if the surrounding road is not adequate.

3.2 Data preprocessing and analysis

This section presents further information about the main feature set and describes the decisions that were taken when preparing the data. Our choices for the data preprocessing such as handling outliers, feature scaling, and missing data will be discussed in subsections 3.2.1 and 3.2.2.

	min	mean	median	std	max	availability
L. Geo	-224.54	-0.63	-0.15	1.55	0	0.99
R. Geo	-363.51	-1.60	-0.25	2.95	0	0.99
L. Type	-8.58	-3.18	-0.15	3.28	-0.12	0.93
R. Type	-8.58	-3.39	-0.15	3.92	-0.12	0.79
L2. Geo	-308.59	-4.17	-1.91	3.54	0	0.90
R2. Geo	-399.26	-5.49	-7.67	3.84	0	0.46

Table 3.1: Table of metrics for the primary feature set. The minimum, mean, standard deviation, maximum and proportion of present values are shown for each feature in the main feature set (Model Qualities).

Table 3.1 shows descriptive statistics of the feature set, including minimum, mean, median, standard deviation, maximum and availability for each feature.

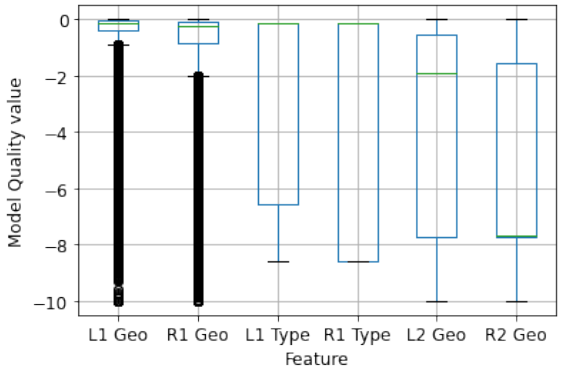


Figure 3.4: Boxplot showing a box of Q1-Q3, the median value as a green horizontal line and whiskers reaching at the most 1.5 times the length of the box. Outliers are presented as circles.

3.2.1 Outliers and Scaling

We take outliers to be values less than -10. These values far from the mean may occur due to variance or it may be due to some anomaly.

A lower bound is placed on the features at -10, since the information given by any value lower than that is not deemed to add any additional information. This is because any model quality value that is very low is likely to be due to some error in the preparation of the data. This choice is also made since some machine learning models (e.g. linear models) may be affected by outliers more than others, such as decision tree ensembles.

Table 3.1 shows that outliers are present for the geometrical features.

A common approach for feature scaling is to subtract by the mean and divide by the variance. The reason for this is that some machine learning algorithms use the distance between data-points to perform clustering or classification, and if the scale of the features differ significantly, features with larger values will dominate the outcome. The MQs differ slightly in their means and variances, but not by a lot. Therefore, we have chosen to not perform any scaling, normalization, or standardization on the data.

3.2.2 Missing data

Missing data can be handled, for example, by removing any rows in which some data is missing, or imputing values to fill in any missing values. Both of these methods have their drawbacks. Removing missing data will reduce the number of available data points for training. It also runs the risk of removing data disproportionately if some scenarios more commonly produce incomplete data. Imputation can be done, for example, by adding the mean of the feature in any place where the feature is missing. There are no guarantees that this will work well as any imputation of data means introducing something that was not present originally.

Several factors can affect the availability of a certain feature. The corresponding element needs to both be perceived by the sensors and be present in the HD-map. For example, to have a primary lane marker geometry measurement, it means that a lane has been perceived by the sensors, this lane has been found to exist in the map, and a similarity measurement has been computed between the perceived lane and the lane in the HD-map. If any of these steps are not completed, the measurement will not exist for that time step.

In table 3.1, the availability for each feature is presented. Primary lane marker geometries are available in 99% of the data set, but for the right lane marker type and especially the secondary right lane marker geometry, availability is limited to 79% and 46% respectively.

The problem with removing incomplete data is that our data set contains a lot of rows with at least some missing data, which means that the size of the training set

would be drastically reduced. A lot of valuable data would also be discarded, as there is no reason to assume that all features are necessary to discern between a true and false label. Furthermore, in a prediction scenario, the availability of data will probably reflect the availability in the training data set.

In our case, examples for imputing missing values can be done with the mean, the median, or the mode of the data. This prevents the loss of data from deleting values, but the value that is inserted for a given row may not be appropriate. This is because this imputation method does not take into account the co-variance of features. The exact methods for imputation will depend on the experiments being done and will be discussed in more detail in the "Experiments and results" chapter.

3.2.3 Data visualisations

This section presents visualization of the data such as pairwise correlations and descriptions and distributions of the features split on the target label made with the aim of better understanding the data set.

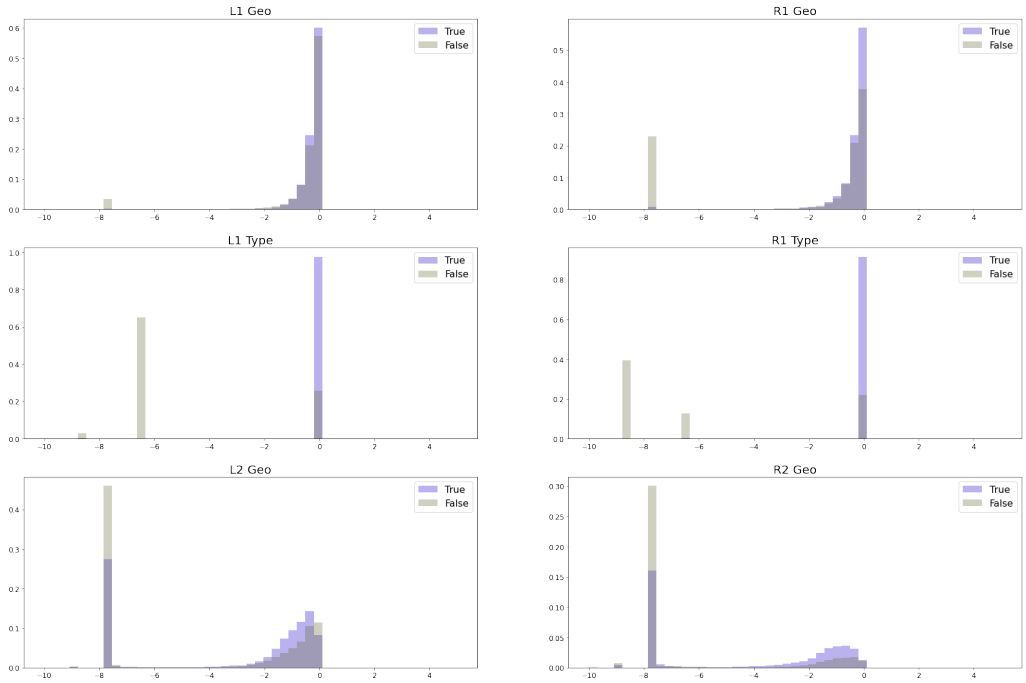


Figure 3.5: Momentary MQs for samples coming from correct and false hypotheses respectively. The distributions are normalized to show density, i.e. the total area of each of the two distributions sums to 1 (y axis). On the x axis is the MQ-value. The distributions are presented by a histogram of 50 bins over [-10, 5] and the data is split into two colors by label.

Figure 3.5 shows the feature distributions and table 3.2 presents the mean and standard deviation of the MQs when split on the target label. From table 3.2, it can be noted that lane marker type qualities takes discrete values. This is because

	Feature - mean(std)					
Hypothesis label	L1 Geo	R1 Geo	L1 Type	R1 Type	L2 Geo	R2 Geo
True	-0.30(0.59)	-0.41(0.88)	-0.19(0.58)	-0.16(0.50)	-3.04(3.20)	-4.11(3.24)
False	-0.59(1.48)	-2.13(3.26)	-4.88(2.93)	-5.73(3.70)	-4.37(3.53)	-6.17(3.90)

Table 3.2: Table showing the mean and standard deviation of the data set when split on the target label, meaning if a given row is produced by a true or false hypothesis.

there are only a certain number of possible lane marker types, and therefore also only a certain number of possible marker type combinations.

False hypotheses produce lower MQ averages with higher variance than true hypotheses. Since the Model Qualities may contain more or less noise coming from sensor measurements, inaccurate associations, or inaccurate HD-map, it is of interest to explore if the features can be preprocessed in some way to increase the performance of a probabilistic classifier.

Given the numbers from table 3.2, averaging MQs may be one way to easier be able to discern if a row comes from a true or a false hypothesis. Figure 3.7 shows the distribution of the mean feature values over whole sequences and for true and false hypotheses respectively. Indeed, the feature distributions for the different labels seem to be more separable than in 3.5.

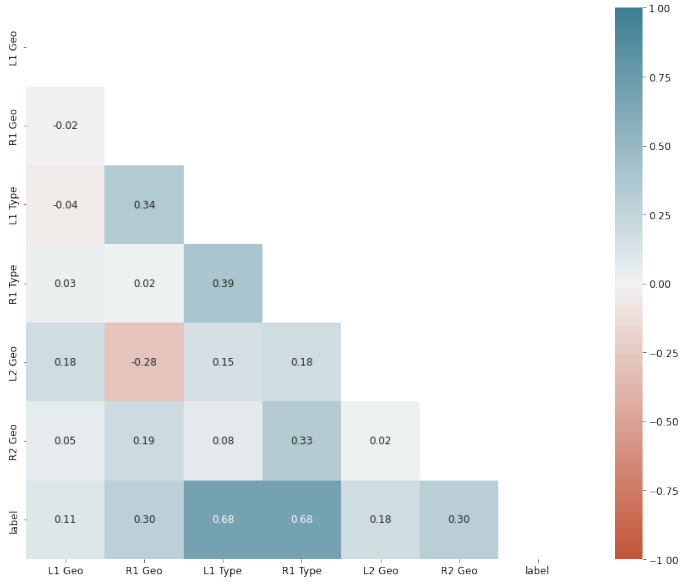


Figure 3.6: Pairwise correlation matrix of the features and the target label using Pearson correlation coefficient. A more blue color means a stronger positive correlation while a more red color means a stronger negative correlation. A value closer to 0 or a lighter color closer to white means no correlation. Any absolute correlation value below 0.5 is considered more or less weak.

Figure 3.6 confirms what has been seen in the distributions and table 3.2 and tells

us that the "lane marker type" MQs have a strong correlation with the target label.

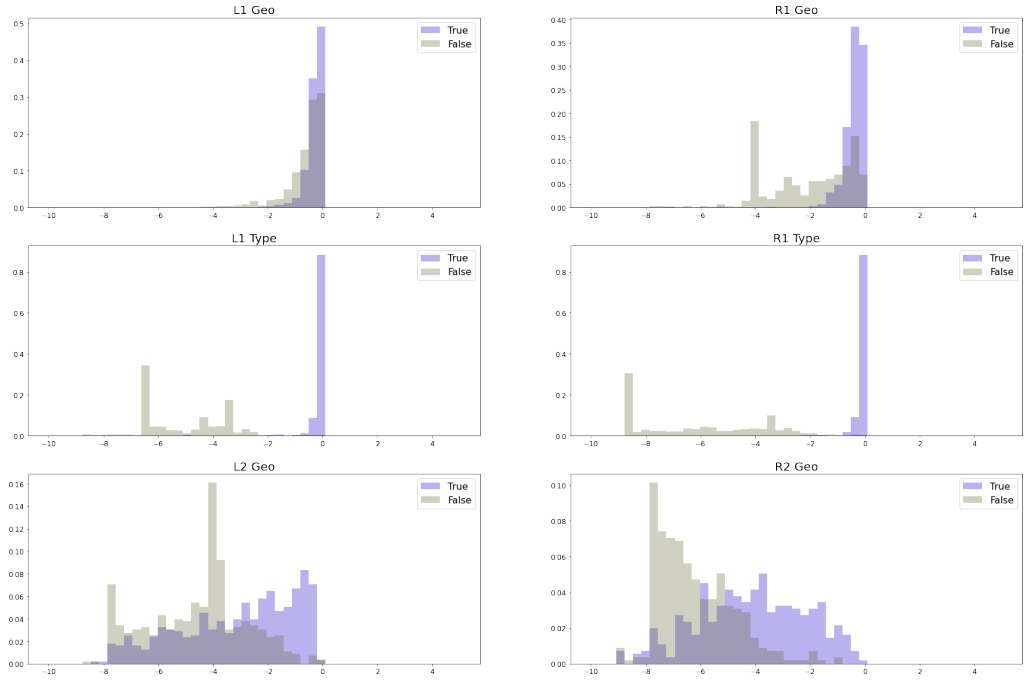


Figure 3.7: Averaged version of the feature set, 50 bins over [-10, 5]. Color by label. The average is taken as the mean value for each feature over whole sequences.

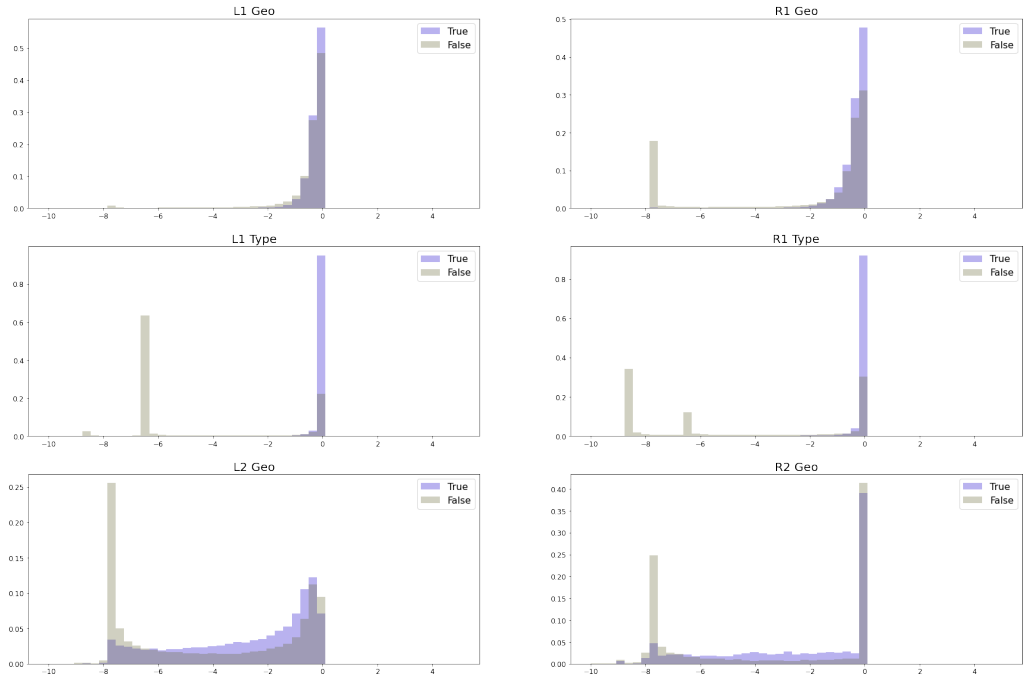


Figure 3.8: Low-Pass version of the feature set, 50 bins over [-10, 5]. Color by label.

In Figures 3.7 and 3.8, the difference in distribution can be seen using two different

methods for introducing a temporal component.

For a correct hypothesis, the averaged Model Qualities are expected to converge to better values as time passes and more samples are collected. This is expected since it is assumed a correct hypothesis will continue to be correct. For an incorrect hypothesis, the values are expected to converge to poorer values. Good values may occur in the momentary features for an incorrect hypothesis if there is similarity in the types or the geometries of the lanes between the incorrect hypothesis and the correct hypothesis. In these cases the similarity causes the model to be unable to discriminate between the hypotheses.

Another reason for good values for an incorrect hypothesis is that something has occurred during the collection or processing of the data. An example of this is the vision system incorrectly identifying a solid lane marker as a dashed lane marker due to wear on the road. Problems such as these may occur for only a short period of time, and such problems with the data can be bridged by averaging the features.

Together with the insights from figure 3.7 and table 3.2, this suggests that a good idea would be to use a moving average as input for classification, as opposed to the momentary MQs. The idea is that, over time, a correct hypothesis will have feature values converging to better values, making it easier to identify as a correct hypothesis. Using an average is also a means of imputing the data that is missing in the momentary model qualities.

3.3 Data split

When evaluating the performance of a model it is important that it is tested on data which it has not been trained on. Therefore, it is customary to split data into separate training- and validation data sets.

Since a model's performance will reflect what it has encountered in its training, it is of interest to let the training data be as varied as possible. The sequences are therefore evenly split around the city of Gothenburg. In figure 3.9 the data set splits are shown on a map.

	(1) Green	(2) Blue	(3) Red
No. sequences	88	104	361
L1 Geo	-0.359 (1.025)	-0.655 (1.416)	-0.874 (1.751)
R1 Geo	-1.72 (2.970)	-1.052 (2.436)	-1.326 (2.448)
L1 Type	-3.352 (3.241)	-3.159 (3.231)	-2.928 (3.409)
R1 Type	-3.959 (4.060)	-2.827 (3.698)	-2.843 (3.725)
L2 Geo	-3.505 (3.454)	-4.846 (3.302)	-4.999 (3.367)
R2 Geo	-5.412 (3.396)	-6.095 (3.876)	-5.074 (4.481)

Table 3.3: Table showing the number of sequences included in each data set as well as the mean and standard deviation for each feature.

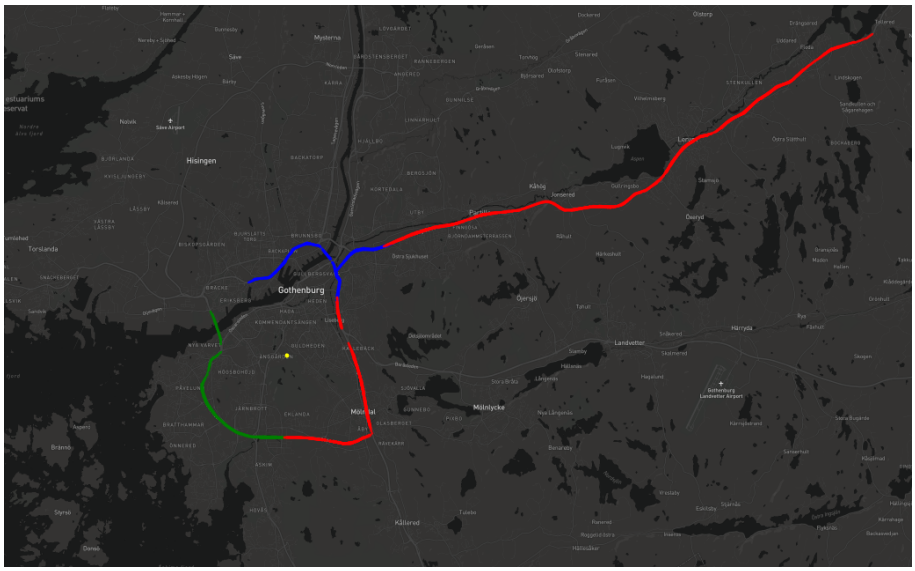


Figure 3.9: Map showing the three different data sets in the colors red, blue and green. They have been split into different sets based on the angle from the reference point marked with yellow and their mean position.

There needs to be multiple sets as training, validation, and testing will be performed. This split has been chosen to be done geographically so each set covers distinct areas of Gothenburg.

3.4 Annotation

Creating high-quality data annotation is a tricky topic. One approach is to use manual annotation of data with agreement. This means that all items in the data set are shown to someone and each item is assign a class by that person. This can be done using agreement between multiple people for a higher quality annotation.

Another approach is to use some form of automated annotation. This can be employed for tasks where some information relating to the ground truth of the data is available during training, but not during usage of the finished model. This extra

information can then be used to create ground truth labels for training the model.

Automated annotation is more scalable than manual annotation but the quality of the labels will depend on the annotation procedure. For this thesis, there is no possibility of creating manual annotations as it too laborious. An annotation procedure has therefore been developed.

The data annotation is done automatically based on the reference model of the vehicle state. The longitudinal component is not considered since the main goal of the model is to localize the vehicle at the lane level. A fundamental assumption is that at any point in time there is at least one hypothesis which is correct.

The annotation procedure requires is a minimum distance to the reference model for labeling an hypothesis as correct, as well as a window for ambiguity that is used in the labeling algorithm. Any hypothesis that is further away from the ground truth than the minimum distance cannot be labeled as correct. In the case where at least one hypothesis is within the minimum labeling distance, and another hypothesis is within the window of ambiguity, no hypothesis will be labeled as correct for that time step.

Multiple hypotheses can be labeled as correct if they both are within the minimum distance for labelling, and in this case we define these hypotheses as *converged*. This labelling is done for every time step (40Hz). In other words, one or more hypotheses can be labeled as correct if no other hypotheses are close enough for the classification to be unclear.

The procedure for annotation is as follows:

The pose of the reference vehicle is extracted as geodetic coordinates and the positions of the hypotheses are converted into a Cartesian frame with origin at the position of the reference vehicle. For each hypothesis, it's position in x is the longitudinal offset of an hypothesis with respect to the reference vehicle, and the position in y is the lateral offset.

The hypothesis with the smallest y is considered the best hypothesis.

Around six percent of the whole data set lacks labels. The missing labels represent the time steps at which there is ambiguity as to which hypothesis is correct.

Sequences with missing labels are removed from the data set as the labelling for these sequences are judged to more likely be of lower quality.

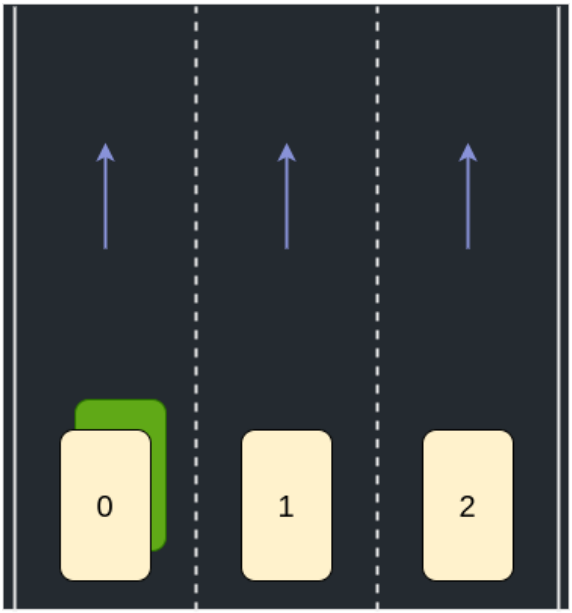
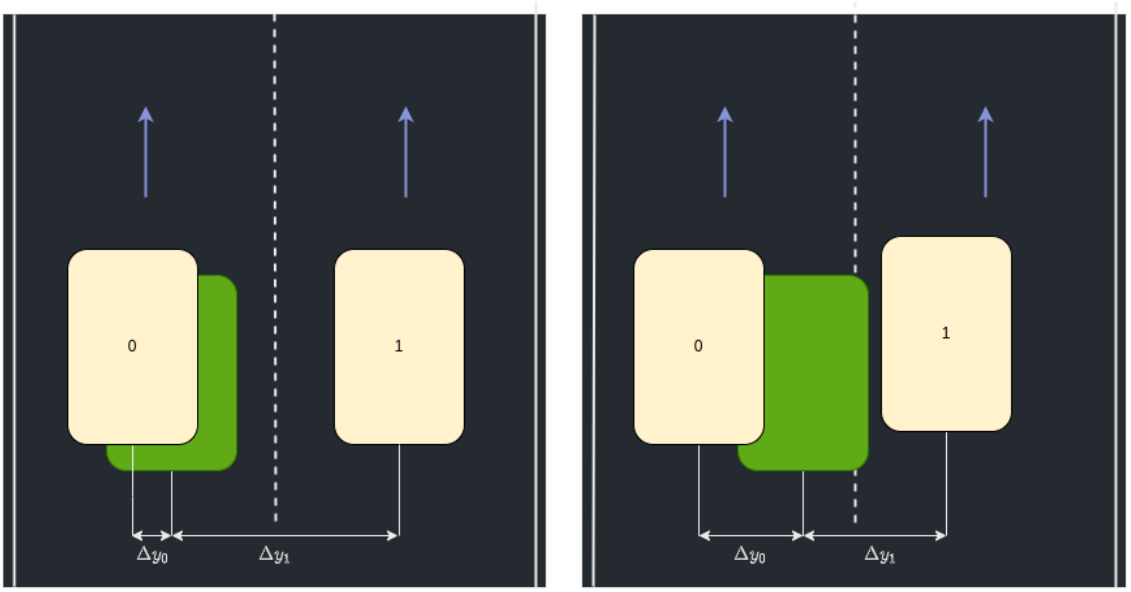


Figure 3.10: Example annotation. The white shapes show the pose of the possible hypotheses and the green shape show the pose of the reference vehicle. The arrows signify lane center line and direction of movement.

In Figure 3.10 there are three active hypotheses (white boxes). Hypothesis 0 is very close to the reference vehicle, and for this time-step, we would like the annotation to reflect that this must be the correct hypothesis.



(a) (b)

Figure 3.11: Two annotation scenarios. In the left scenario, $\Delta y_0 << \Delta y_1$ and hypothesis 0 is annotated as the correct hypothesis. In the right scenario, y_0 and y_1 are similar, and no annotation is made for that frame.

Figure 3.11 shows two annotation scenarios on a straight road. When the reference vehicle is close to the center, annotation becomes more difficult. That the reference vehicle is close to the center may be due to driving close to the center line, but also due to some problem with the reference system or the overlaying of the reference on top of the HD-map.

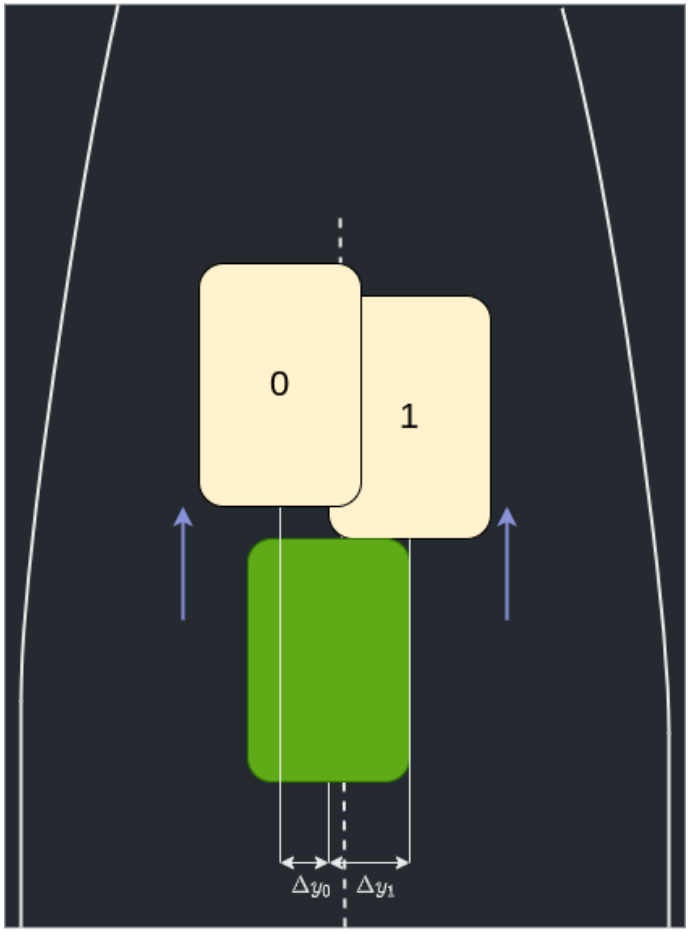


Figure 3.12: Example of lane merge leading to a convergence of hypotheses and ambiguity in annotation. Reference vehicle is drawn behind the hypotheses for clarity, as longitudinal offset is not considered for annotation purposes.

In cases where hypotheses converge, like during a lane merge as seen in Figure 3.12, the hypotheses will end up overlapping. If there is no ambiguity in the annotation process before the merge, there will be one true label before the merge, no true label during the merge, and two true labels after the merge. As the task is to perform Lane-Level Localization, an additional hypothesis within a lane does not contribute with more information. For this reason, whenever an hypothesis converges into the true hypothesis, that hypothesis is removed from the timestamp of convergence.

The same thresholds that are used for labelling are used for defining convergence.

For ensuring that the annotations and the data are of good quality, the sequences

are required to be fully annotated.

4

Method

The various aspects of our methodology are presented. This is followed by our definition of a time series in section 4.2 and a description of our methods for developing our solution in section 4.3. A description of the evaluation strategy is presented in 4.4.

4.1 Experimental setup

All programming is done in Python and, where nothing else is noted, the library *scikit-learn* (Pedregosa et al., 2011) is used for the building and implementation of the different machine learning models.

To be able to train a supervised machine learning model, ground truth labels must be present. These ground truth labels will be calculated from a reference model. The reference model is calculated offline by using data from a set of external sensors with high precision during data collection, including an OXTS system.[1]

As the reference model is derived from processed sensor measurements it cannot be guaranteed to be correct, but it will be more accurate than the online systems.

4.2 Definition of a multivariate time series

Time Series (TS) and *Multivariate Time Series* (MTS) are defined as follows, based on Xing et al. (2012) and Mori et al. (2019), but adapted to our purpose:

A *time series* (TS) is a time-ordered sequence of pairs (timestamp, value) with some maximum length L.

$$TS = \{(t_i, x_i), i = 1, ..., L\} \tag{4.1}$$

Where x_i is the value of TS at time t_i. x_i is also referred to as the *momentary value* of TS at i.

[1] Oxford Technical Solutions. An intertial navigation system (INS).

A *multivariate time series* (MTS) is a time series in which the values are vectors such that $\boldsymbol{x}_i \in \mathbb{R}^n$ where n is the number of values of the time series.

$$MTS = \{(t_i, \boldsymbol{x}_i), i = 1, ..., L\} \tag{4.2}$$

The evaluation of a (multivariate) time series at a certain time step is also defined as follows: Let $MTS|_t$ be the *prefix* of MTS at t such that

$$MTS|_t = \{(t_i, \boldsymbol{x}_i),\ \forall\, i \ \text{ where } t_i <= t\} \tag{4.3}$$

4.3 Solution design

The problem is further specified as finding a method for classification of time series that is *early* and *accurate*. Beyond this, the solution should be scalable, meaning it should work well for more data, longer lengths of time series, more features, and more hypotheses. It should also be robust to poor data quality and only produce predictions with a low false positive rate.

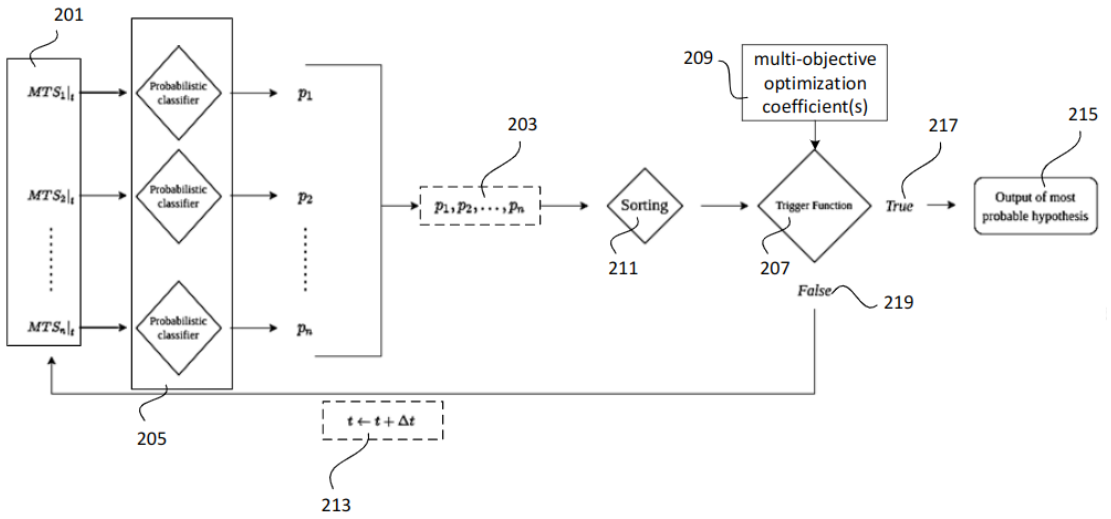


Figure 4.1: An overview of the full solution architecture, based on the process in Mori et al. (2019). $MTS|_t^n$ is the n:th time series prefix, up to time t. p_k is the predicted true-probability as given by the probabilistic classifier for hypothesis k. A trigger output of 1 signifies an output of the hypothesis corresponding to the highest probability in P.

The solution design for this project was based on the solution that was presented in Mori et al. (2019) and adapted to our specific goals and domain. The full architecture of the Hypothesis Inference solution is shown in figure 4.1. The input of the model is a set X of n time series, evaluated at time t. The output of the model is the index of the hypothesis with the highest probability of being correct.

The solution consists of two steps. First, the multivariate time series data is input into a probabilistic classifier. This classifier is binary, where the "true" class signifies

that an hypothesis is correct, and the "false" class signifies that an hypothesis is incorrect. The probabilistic output is the belief of the classifier that an hypothesis is part of the "true" class.

A comparison of the probabilistic outputs is what is used to determine the certitude with which a selection can be made. The outputs are sorted by magnitude and are used as input to a trigger function whose output determines if a selection should be made at this time step or if the model needs more data (time) to become more confident in that the highest probable hypothesis, given by the probabilistic classifier, is correct.

The architecture contains two primary parts: the probabilistic classifier and the trigger function. The following chapter describes the results in developing these two parts, as well as the results of the unified solution.

Some limitations for the design of the probabilistic classifier and the trigger functions follow:

The probabilistic classifier has been chosen to make its first output at time step 40. This corresponds to one second at 40Hz and has been chosen for comparability between the various methods. It could be set to a different value, but a lower value increases the likelihood of no data being present up until that point.

As some methods require input data of specific length, a set of evaluation points (time steps) have been chosen at which the probabilistic classifier evaluates the sequences. Six evaluation points have been chosen on a logarithmic space between 40 and 1201. These choices have been made partly to facilitate a more fair comparison for different probabilistic classifiers, but also has the added benefit of making the algorithm faster in both training and testing.

4.4 Evaluation

The primary scoring metric used is the accuracy which is a very common classification metric because of its simplicity and interpretation. It corresponds to the ratio given by taking the sum of true positives and true negatives divided by all predictions, or in simpler terms, the percentage of correct predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

The performance of the genetic algorithm in the multi objective optimization will be evaluated with the help of the hypervolume with a reference point set to [1.1,1.2]. This point is chosen arbitrarily and is slightly worse than the theoretically worst possible cost of the two objectives. Since the multi objective optimization will contain also availability and earliness, these two measures will also be evaluation metrics in addition to accuracy. Earliness is defined as the average portion of a sequence which has passed before a prediction is triggered. Availability is a sort of

discretized earliness measure, in which a prediction will be defined available if it is triggered during the first 30 seconds of a sequence (before it is finished).

An *Achievement Scalarization Function* (ASF) will return an optimal solution based on a vector of weights, in this case the weights are set to [0.2,0.8] for accuracy and availability, as well as accuracy and earliness. The weights are set arbitrarily but with the intent that accuracy is more important than availability/earliness. Put shortly, the reason for favoring accuracy is because the localization output is used by other systems in which it is more interesting to be correct than to be fast in prediction.

For evaluation of the final solution, both quantitative and qualitative measures will be adopted. Interesting cases that will be evaluated include sequences in which a wrong hypothesis is selected, sequences with a late prediction and sequences in which the trigger function does not trigger a prediction. The qualitative evaluation consists of choosing a representative sequence, showing the selected plots of data and discussing the scenario in relation to the inference method.

5

Experiments and results

This chapter explains and motivates the approach by breaking the problem down into smaller components. The main components of the solution are presented in sections 5.1 and 5.2. These sections contain intermediary results and evaluations. A presentation and discussion of the performance of the solution follows in chapter 6. The results will be presented in a way that highlights the process and will therefore include parts which are commonly seen included in the method.

5.1 Probabilistic classifier

This section deals with finding a probabilistic classifier for the multivariate time series data.

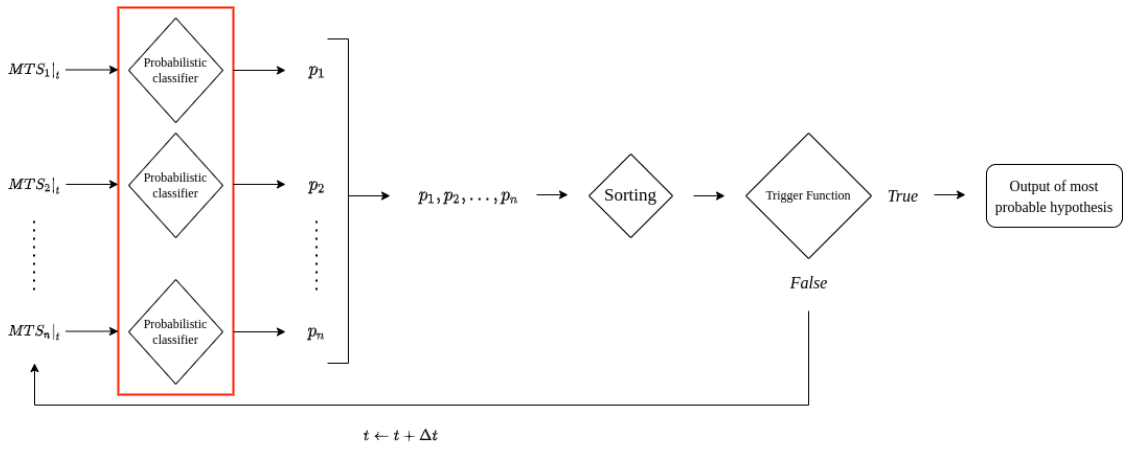


Figure 5.1: Solution architecture with the probabilistic classifier highlighted.

We considered using a deep learning approach but chose not to include this as it was deemed to take a lot of time and energy which could better be directed elsewhere. We did not believe an LSTM- or CNN-based solution would be significantly better than simpler models due to the low dimension of the data.

It was realised early that there would be an advantage to use a probabilistic solution that is not dependent on exact lengths of time series. This would allow for continuous

output of belief in hypotheses instead of specifying a set of time steps to be used for evaluation. This makes the model more reactive as output does not have to wait until the next evaluation step.

Using models that are not dependent on input size will also make both the training and the usage of the models simpler to manage. The reason for this is that a single model can be used for all the time steps instead of training a model for each evaluation step. This reduces the total complexity of the solution and makes it more scalable. For our purposes, models that require a fixed input size could be used, as the time space is capped at 30 seconds (1200 time steps), but for a real-time processing scenario, the length of driving sequences are not restricted to 30 seconds. As was noted by Schäfer and Leser (2020), the longer the sequence runs, the less data will be available for training and for evaluation. This problem is also avoided by using a model that is not dependent on input size.

There has been a lot of work done in the classification of time series and we wanted to include some state-of-the-art method for comparison purposes and to see if it would actually perform better. This is presented in section 5.1.4.

5.1.1 Momentary measurement model qualities

In this section a selection of machine learning algorithms are tested on momentary data. This is the data that has not been averaged or filtered over time. The measurement model qualities are computed solely on measurements acquired at the current time step. The results will be used as a heuristic to which model to be used for further development. The following set of machine learning classifiers are compared:

- Gaussian Naive Bayes, NB
- Logistic Regression, LR
- Linear Support Vector Classifier, Linear SVC
- Kernel Support Vector Classifier, Kernel SVC
- Random Forest Classifier, RF
- Gradient Boosting Classifier, GB

Naive Bayes, Logistic Regression and Linear Support Vector Machines are simple, widely used models which are quite easy to understand. The Kernel SVM is chosen as a non-linear SVM alternative, using a radial basis function kernel. Ensembles of decision trees are popular and often well performing for data which is in dense tabular form. Therefore, a Random Forest classifier and a Gradient Boosting classifier are added to the model comparison.

All above models are trained in their default configurations as presented in scikit-

learn. The motive behind not tuning the models is that the default configuration is deemed good enough at showing how well a model performs on a given data set while at the same time keeping the complexity of the task down.

The model selection experiment will be run in two distinct scenarios which differs only in how missing values are treated: In scenario 1, the missing data is imputed by a constant, 5.0. This is the original form of the data as presented to us. In the second scenario, the missing values are imputed with their mean value.

To make the comparison more informative, the models will be trained on seven data sets of different sizes, where the sizes are spread evenly on a logarithmic scale from 500 to 70000. The rows are sampled at random from the training data set but with a set seed value for reproducibility and consistency. Given the nature of the data being quite similar between adjacent time steps, we expect the random sampling to provide us with a good representation of the full data set while keeping the cost of training time down.

Each probabilistic classifier will be run and evaluated by a 5-fold cross validation score, i.e. the training data is split into 5 folds and in each iteration the classifier use one of the folds as validation set while training on the other four folds. By using the *GroupKFold*-class in scikit-learn, the folds will be sampled on sequences, hence no validation set will include samples from a sequence seen in that particular training.

The performance of the models is evaluated on their accuracy score. Also, feature importances from the Gradient boosting tree ensemble are extracted and presented.

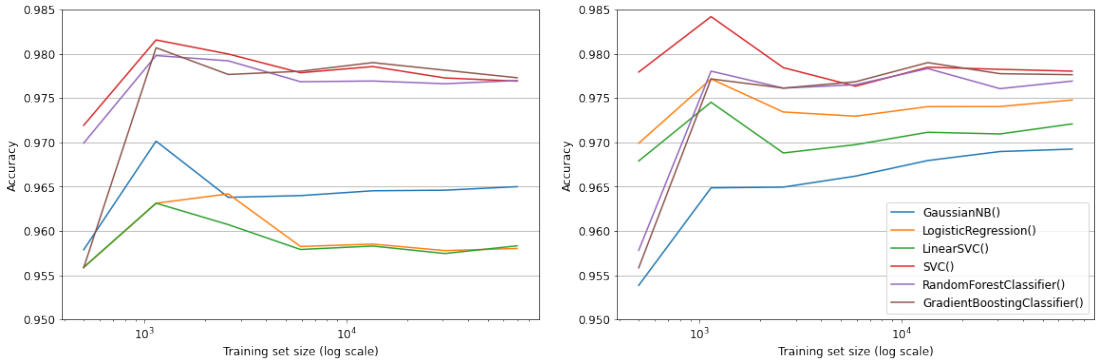


Figure 5.2: Line plot showing the accuracy for different classifiers and amount of training data. The models in the left figure are trained and tested on momentary MQs with missing values replaced with the constant 5. The models in the right figure are trained and tested on momentary MQs with missing values replaced with the mean value for the given feature in the training set.

Figure 5.2 presents the results from the 5-fold cross validation scoring. The left figure shows when missing values are replaced with the constant 5. Naive Bayes and the linear models perform in the 0.96-area of accuracy while the decision tree ensembles and the non linear SVM reaches an accuracy of around 0.98. The figure to the right

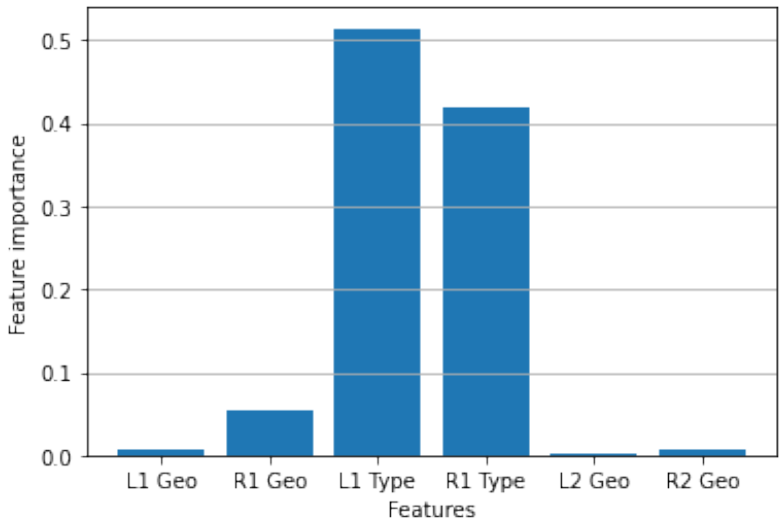


Figure 5.3: Bar plot showing feature importances extracted from the trained Gradient Boosting classifier. All values sum up to 1.

presents the results from the same experiment but with missing values are replaced with the mean for that feature. I.e. the mean value for each and every feature in the training set is used to replace missing values in the training- and validation sets. This feature imputation technique clearly increased the performance for the linear models, but did not significantly change the performance for the other models. As long as the training size is at least 1000, adding more data to training do not seem to affect the accuracy scores significantly.

Figure 5.3 presents the feature importances extracted from the fitted Gradient Boosting classifier. It clearly shows that the marker types are the features which the model relies on the most for prediction. Note also that R! Geo has a higher feature importance than L1 Geo. Table 3.1 shows that there is a difference in the distribution of R1 Geo between the true and false labels.

5.1.2 Averages

This subsection presents experiments in which the time component of the data set will be considered in training and validation. To decrease the complexity of the experiments, only the Gradient Boosting classifier will be trained and validated on the different scenarios. This classifier has been chosen since it performed well in the previous experiments. Also, being based on decision trees, it is quite robust with regard to the existence of missing values.

The choice of constructing input data to the classifier which encompass temporal information is because the momentary measurements contain more or less noise. Also, a correct hypothesis should over time converge towards having better averages than a false hypothesis, given that the situation would be one that could theoretically be distinguished by the sensor measurements, i.e. lane marker geometries or lane marker types that differs between a correct and a false hypothesis. Therefore, for

the following experiments, the models will predict on the current average for each feature and time step.

A model which predicts on the current average will probably perform differently if it is trained on momentary values as opposed to some kind of averaged values. This will be evaluated below. Another aim of this experiment is to note if the performance increases with a passed number of time steps as the model predicts on the current average for each feature.

The first experiment will consist of a 5-fold cross validation where a Gradient Boosting classifiers will be trained in different ways, either with momentary values, moving averaged values (window sizes of 10, 100, 200 or 400 time steps), or with a sequence average (one sample per sequence and hypothesis, containing the mean values of the features).

The classifiers trained on momentary values and moving averaged values will be trained on 10000 samples while the model trained on sequence averages trains on 813 samples since this is the number of time series in the training set. The models predict on each time step and the accuracy measurements for different time steps and classifiers are presented in figure 5.4. The figure legend describes on which data the models have been trained on. It is clear that the accuracy measurements are improving with the number of passed time steps for each model. Models trained on a rolling average of size 200-400 seem to achieve the best accuracy.

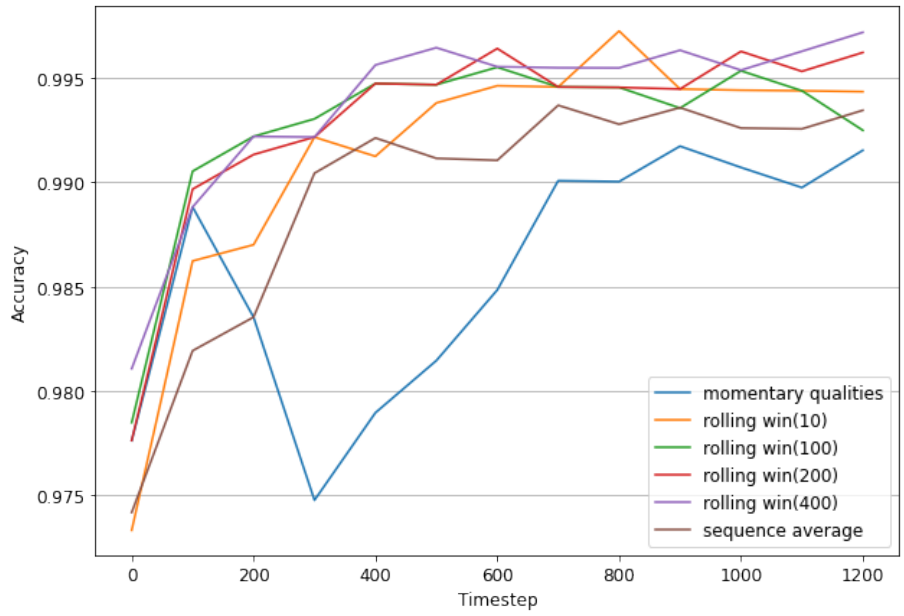


Figure 5.4: Line plot showing the 5-fold cross validation accuracy for different time steps and differently trained GB classifiers, averaged over ten runs. The legend describes what each GB classifier has been trained on. Numbers in parentheses describe the size of the moving window, over which it averages.

5.1.3 Constructed feature set

Classification on average qualities can be extended to include more features than just the averages of the original columns. In this section, we construct a feature set based on our knowledge of the domain.

The type information for the lane markers have been shown to be of high importance for classification. From a practical perspective, this seems reasonable as mismatched geometry may be due to some small miss-alignments or imperfect precision in vision, but mismatched marker types should be a strong indicator that the hypothesis is incorrect. Lane marker type information has been incorporated in localization systems by Cui et al. (2016) and Lee et al. (2021).

The constructed feature set will contain the following information:

- Mean of MQs
- Standard deviation of MQs
- Harmonic mean of average type MQs
- Availability of type qualities

Mean of MQs is calculated by taking the mean of each feature for a sequence, and averaging over all features (i.e. a mean of means). This will provide condensed information about the tendency for good MQs in that sequence. As the MQs are measurements of similarity between the perceived lanes and the HD-map, a correct hypothesis is expected to have good MQs on average.

Standard deviation of MQs is calculated by taking the standard deviation of each feature for a sequence, and averaging over all features. This will give an indication of the tendency for MQs to fluctuate. This may be relevant as MQs that have a high variance may be due to sometimes having lane data that matches the correct hypothesis, and sometimes not. A correct hypothesis would be expected to always have matching lane data and therefore a low standard deviation.

Harmonic mean of average type qualities is derived by averaging the left type data and the right type data over the full sequences. The feature is the harmonic mean between these average values. We expect the lane marker type information to be powerful when combined, as a single lane marker type may be correct for an incorrect hypothesis. This approach is a way to combine the type MQs into a single measurement.

$$type\_avg = \frac{S_l^x + S_r^x}{2|x|} \tag{5.1}$$

Where S_l^x is the sum of left marker type MQs and S_r^x is the sum of right marker type MQs.

Availability of type qualities is the portion of rows for which type data is available.

It is calculated by

$$type\_ava(x) = \frac{N_l^x + N_r^x}{2|x|} \tag{5.2}$$

Where x is a sequence, N_l^x is the number of rows in x with left marker type data present, N_r^x is the number of rows in x with right marker type data present, and $|x|$ is the size of x.

This may be relevant as average type MQs can be good even though the actual types does not match. This can happen if one side has a good match, and the other side has no type data present. In this case the average type MQs feature would be close to 0, but the availability would be around 0.5.

It is worth keeping in mind that good lane marker type qualities does not necessarily mean that the hypothesis is correct, as this only means that the lane markers adjacent to the hypothesis have the same types as the lane markers adjacent to the true hypothesis. An example of this is when the road contains more than three lanes and the hypothesis occupies one of the two center lanes. This means that the true hypothesis will share dashed lane marker types with the false hypothesis that occupies the other center lane. In this situation there is ambiguity between the hypotheses, and this will have to be resolved by either incorporating some other information, such as geometrical information, or by waiting until the type ambiguity is resolved.

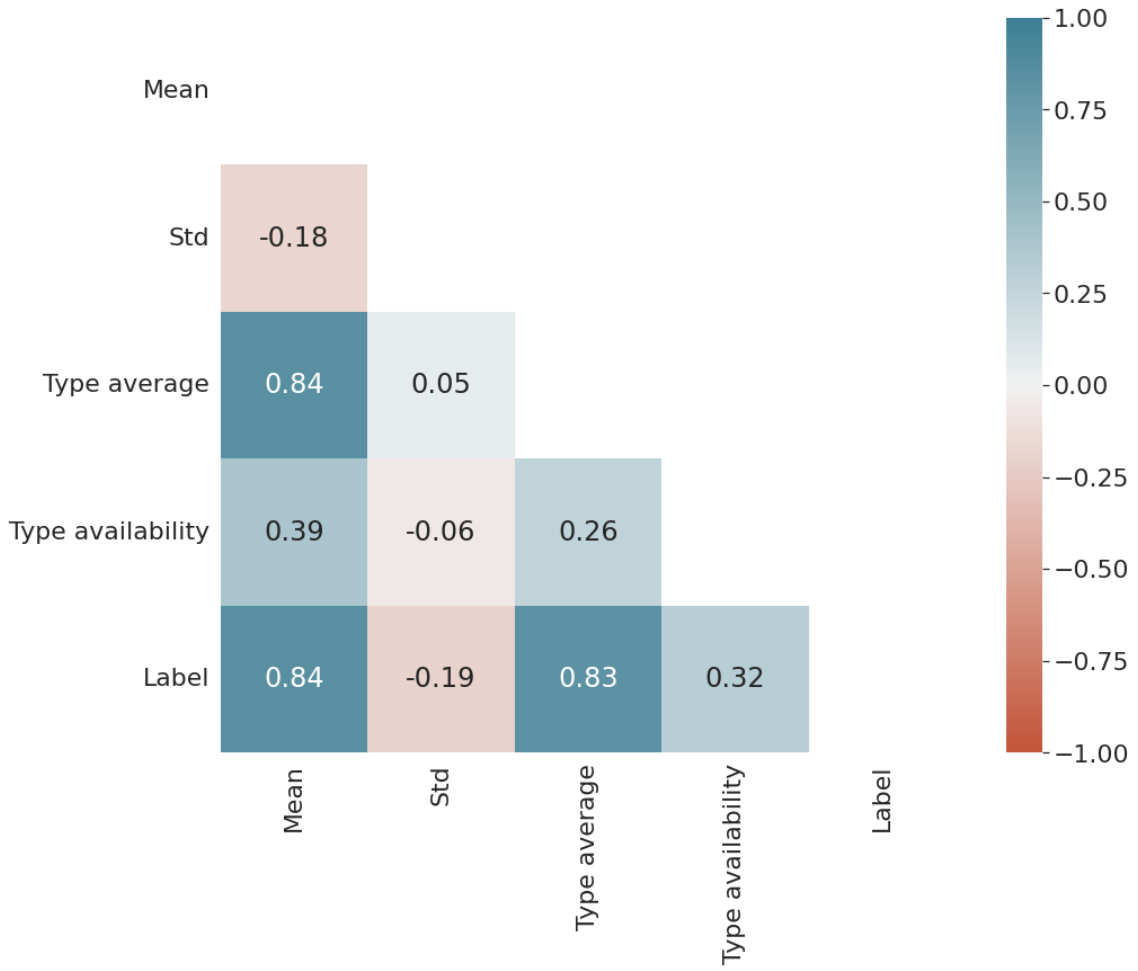


Figure 5.5: Correlation matrix of the feature set as described. A high value means that the features have a positive correlation while a low value means that they have a negative correlation.

The above mentioned constructed feature set is extracted and then used as input to a Gradient Boosting Classifier on the full length sequences (1201 time steps). Testing is done on different lengths of time series prefixes and is evaluated using accuracy score. Validation is performed by 5-fold cross validation using the Green data set. Figure 5.6 presents the cross validation results. The constructed feature set performs well on the training data set. The performance tends to increase as more information is gathered, which confirms our expectation of an earliness-accuracy trade-off.

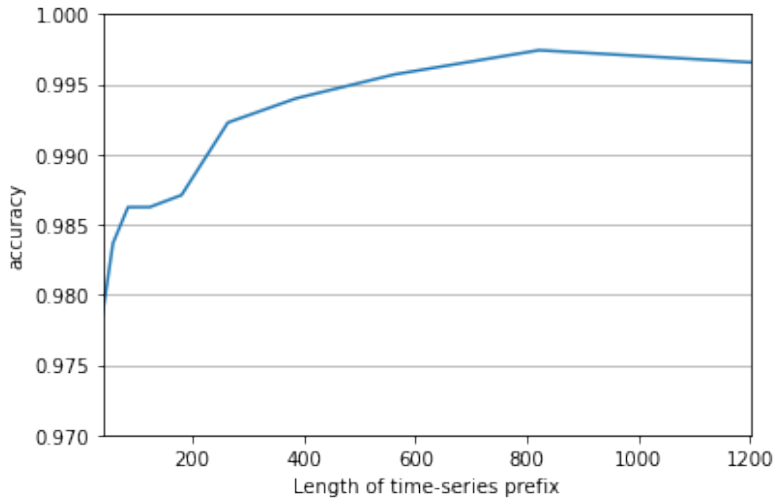


Figure 5.6: Results by training on constructed feature set using a Gradient Boosting classifier with default parameters. Evaluation done on time-series prefix lengths between [40, 1201].

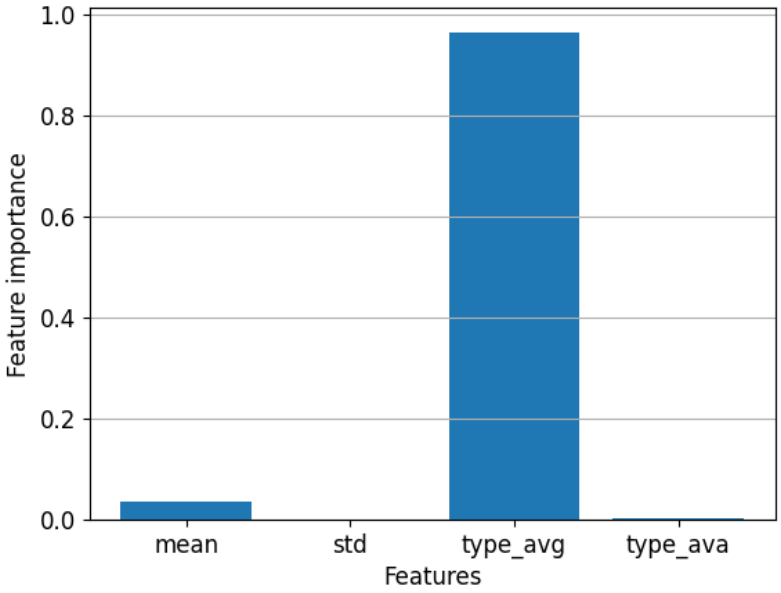


Figure 5.7: Feature importances for the using the Gradient Boosting classifier. All importance-values sum to 1.

Almost all importance is placed on the feature that is derived from the type data. This is in line with what have been seen in previous experiments, that the type data is highly important. Some relevance is also placed on *Mean of MQs*, which shows that the general idea of having low model qualities is associated with a correct hypothesis seems to hold.

The variance of MQs and the availability of type qualities does not influence the classification significantly in this model. Some performance increase would perhaps

be attainable by combining the availability feature with the type averages feature with the idea that a good type average that is based on few samples will be less reliable than a good type average that is based on more samples. On the other hand, the accuracy is very close to 1.0 so in order to document a significant improvement, a more difficult testing set would be necessary.

5.1.4 State-of-the-art methods and candidate selection

Canonical Interval Forest (CIF) will be evaluated as a first tier classifier in the following way: Separate CIF-classifiers will be trained on time series consisting of the first 40,80,160,320,640 and 1200 time steps respectively in each time series. Missing values will be replaced with the constant 5 as in earlier experiments. A 5-fold cross validation-scheme is used on the training set, and the model will be compared to a Gradient Boosting Classifier trained on moving averages with a window size of 200 and predicting on current average for each time step, as well as a Gradient Boosting Classifier trained and tested on the constructed feature set.

The results from the model comparison are presented in figure 5.8. All models seem to perform similarly regarding the accuracy, with the two Gradient Boosting Classifiers having slightly better accuracy than the CIF, reaching a fair bit over 0.99.

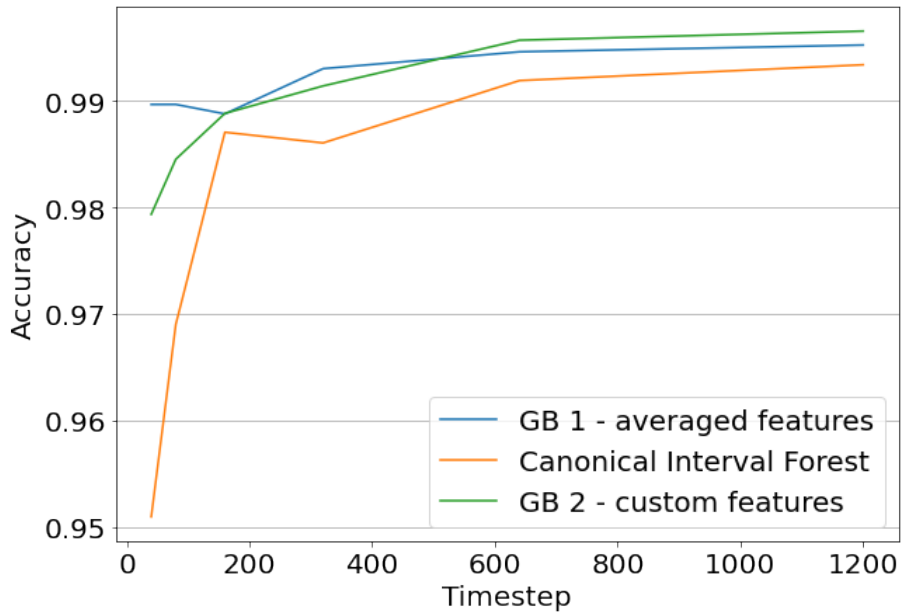


Figure 5.8: Line plot showing the 5-fold cross validation accuracy for CIF-classifiers trained on six different time series sizes compared to a GB classifiers trained on a moving average of 200 and predicting on current average as well as a GB classifier trained and tested on a constructed feature set. The graph shows the accuracy for a given model at a given time step.

5.2 Trigger function

This section deals with the training and selection of different trigger functions. The trigger function is what decides if a prediction will be made for the current time step, or if the model waits for more data before considering outputing a prediction. Figure 5.9 shows in which part of the complete solution the trigger function is. As input, the trigger function takes a vector of probabilities given by the probabilistic classifier, and based on these probabilities it either decides to make a prediction or to wait for more data. The probabilistic classifier is a Gradient Boosting Classifier trained on a moving average of 200 time steps over the MQs. The optimization of the trigger function parameters will be done with regards to two objectives, making it a multi-objective optimization problem which will be approached with the genetic algorithm NSGA2.

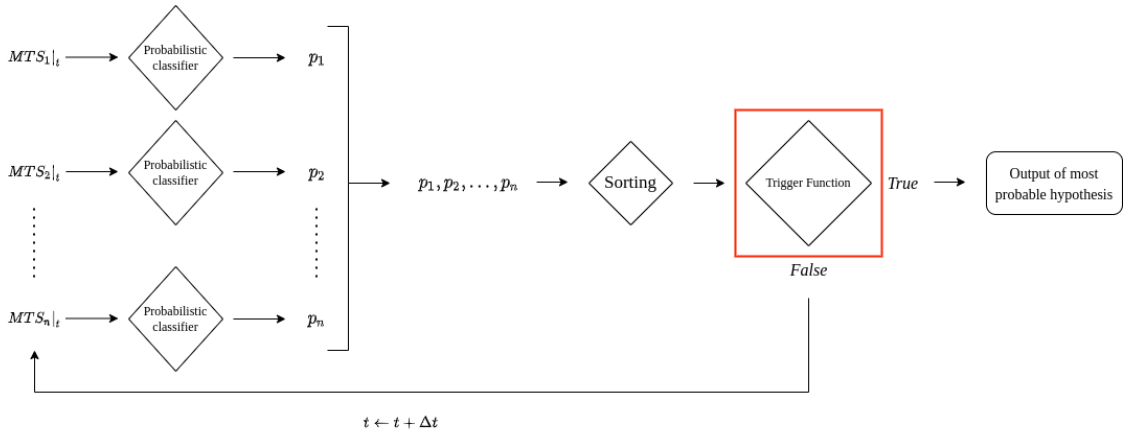


Figure 5.9: Solution architecture highlighting the trigger function.

5.2.1 Multi objective optimization

Two different sets of objectives will be evaluated. As a baseline, the earliness and availability objectives together with a trigger function proposed by Mori et al. (2019) will be optimized. In the other set of objectives, instead of earliness, a cost function for *availability* will be used. The motivation behind optimizing on availability instead of earliness is that for our purposes the availability of a prediction is more important than the earliness of a prediction. In cases where no prediction can be made with certitude, the model should not be pushed to predict by an earliness (time) component. Two different trigger functions will be evaluated for this second set of objectives.

5.2.1.1 Objectives

The earliness- and accuracy objectives are explained in 2. The cost functions for availability is shown in equation 5.3.

$$C_{av}(X, Tr_{\boldsymbol{\gamma}}) = \frac{1}{|X|} \sum_{x \in X} \iota(\nexists \hat{y_x}) \tag{5.3}$$

where X is the set of sequences, $\hat{y_x}$ is the prediction made for sequence x, i.e. the hypothesis with highest posterior probability at time step t_x^* and ι is a Boolean function which takes a value of 1 if the condition is true and 0 otherwise. I.e. ι returns a 1 if no prediction is made for x. Shortly put, the cost function for availability returns the fraction of times when no prediction is made. The explained cost function regarding availability is not that different from the earliness cost function given our scenario, as the sequences are 30 seconds long. I.e. the availability cost is sort of a discretized earliness cost which only outputs a positive value if no prediction has been made during the sequence, while the earliness cost is more continuous in that it returns a growing value from the first time step. As a recap and comparison, the earliness cost function is presented in equation 5.4.

$$C_{ea}(X, Tr_{\boldsymbol{\gamma}}) = \frac{1}{|X|} \sum_{x \in X} \frac{t_x^*}{L_x} \tag{5.4}$$

The first optimization problem is formulated by the minimization of the cost functions earliness and accuracy as in equation 5.5, while the second optimization problem is presented by equation 5.6.

$$\min_{\boldsymbol{\gamma}}(C_{ea}(X, Tr_{\boldsymbol{\gamma}}), C_{ac}(X, Tr_{\boldsymbol{\gamma}})) \tag{5.5}$$

$$\min_{\boldsymbol{\gamma}}(C_{av}(X, Tr_{\boldsymbol{\gamma}}), C_{ac}(X, Tr_{\boldsymbol{\gamma}})) \tag{5.6}$$

5.2.1.2 Optimization algorithm

As in Mori et al. (2019), the algorithm that is used is the NSGA2 algorithm. It is used because of its popularity in multi-objective optimization scenarios. The algorithm has been implemented in python using the *pymoo* package. Except for a parameter change in population size to 8 and termination step of 16, the default parameters are used. The mentioned adjustments to the default parameters has been made to make the algorithm run faster.

The performance of the algorithm is evaluated by use of the hypervolume with reference point [1.1,1.2]. An achievement scalarization function will be used in the selection of one of the solutions from the non dominated solution sets.

The training of the classifier, optimization of the trigger function and testing of the complete model will be done three times, in a rotating manner on the data sets. This is done to be able to test the model on all three different data sets, but still without leaking information from training and optimization to testing. Table 5.1 shows these details. The data sets are presented as colors and can be visualized geographically in figure 5.10. To explain it more in detail, a given classifier will first be trained on the training data set. Then it will be applied to the optimization-data set, in which it will output probabilities for each time step and hypothesis. These probabilities are used in the optimization of the multi objective problem. Lastly, the full solution with the optimized parameters is applied on the testing data set. This is one rotation. The results from the three rotations will be aggregated and presented as a single result for each trigger function.

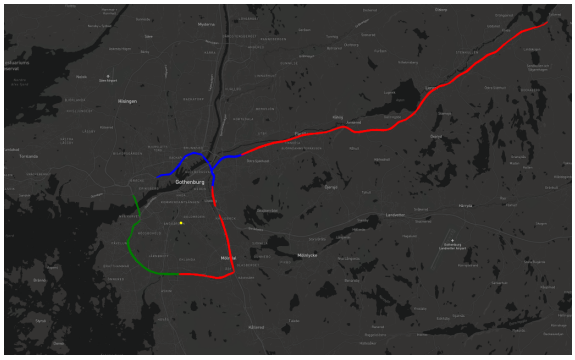


Figure 5.10: Recap of the geographic location of the different data sets. As can be seen, the red data set is largest, with the green and blue data sets being around a third to a half as large as the red.

		Data sets	
Case	Train	Optimize	Test
1	Green	Blue	Red
2	Blue	Red	Green
3	Red	Green	Blue

Table 5.1: Table presenting how the different data sets are used in each of the three experiments. Train means the training of the probabilistic classifier, Optimize is the optimization of the trigger function parameters and Test is the independent data set on which the whole solution is applied.

5.2.1.3 Trigger functions

The following trigger functions will be evaluated:

$$Tr^1_{\boldsymbol{\gamma}}(\mathbf{p}, t) = \begin{cases} 0, & \text{if } \gamma_1 p' + \gamma_2(p' - p'') + \gamma_3 \frac{t}{T} \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{5.7}$$

$$Tr^2_{\boldsymbol{\gamma}}(\mathbf{p}) = \begin{cases} 0, & \text{if } \gamma_1 p' + \gamma_2(p' - p'') \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{5.8}$$

$$Tr^3_{\boldsymbol{\gamma}}(\mathbf{p}) = \begin{cases} 0, & \text{if } \gamma_1 p' + \gamma_2(p' - p'') + \gamma_3 \frac{p'}{p''} + \gamma_4 \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{5.9}$$

where p' and p" are the highest and next highest probability given by the probabilistic classifier, $\boldsymbol{\gamma}$ is the vector of learned parameters, t is the time step for the current prediction and T is the length of the sequence (1200).

The first trigger function, Tr^1, is proposed by Mori et al. (2019) and is being used with the objectives accuracy and earliness. The second and third trigger functions are based on the optimization of the objectives accuracy and availability. Tr^2 is a quite simple trigger function with only two parameters, where only the highest probability and the difference between the highest and the second highest probabilities are considered. Tr^2 is the same trigger function proposed by Mori et al. (2019) but without the time fraction term. The third trigger function, Tr^3, is like the Tr^2 but also includes a ratio of the highest and second highest probabilities as well as a free parameter. This parameter exists to provide a variable threshold for the trade-off between the other terms.

5.2.1.4 Trigger function parameter optimization

In this subsection, the results from the optimization of the multi objective problem will be presented.

5.2.1.4.1 Tr^1 The performance of the genetic algorithm for Tr^1 as well as the non dominated solution sets for the three data sets are presented in figure 5.11. It is clear by looking at the hypervolume graphs that all optimizations converged more or less. A clear pareto front can be seen for the first data set (upper right graph). The red cross shows which solution is chosen by the achievement scalarization function when using the weights (0.2,0.8) for earliness and accuracy respectively. The optimization data sets used can again be seen in table 5.1.

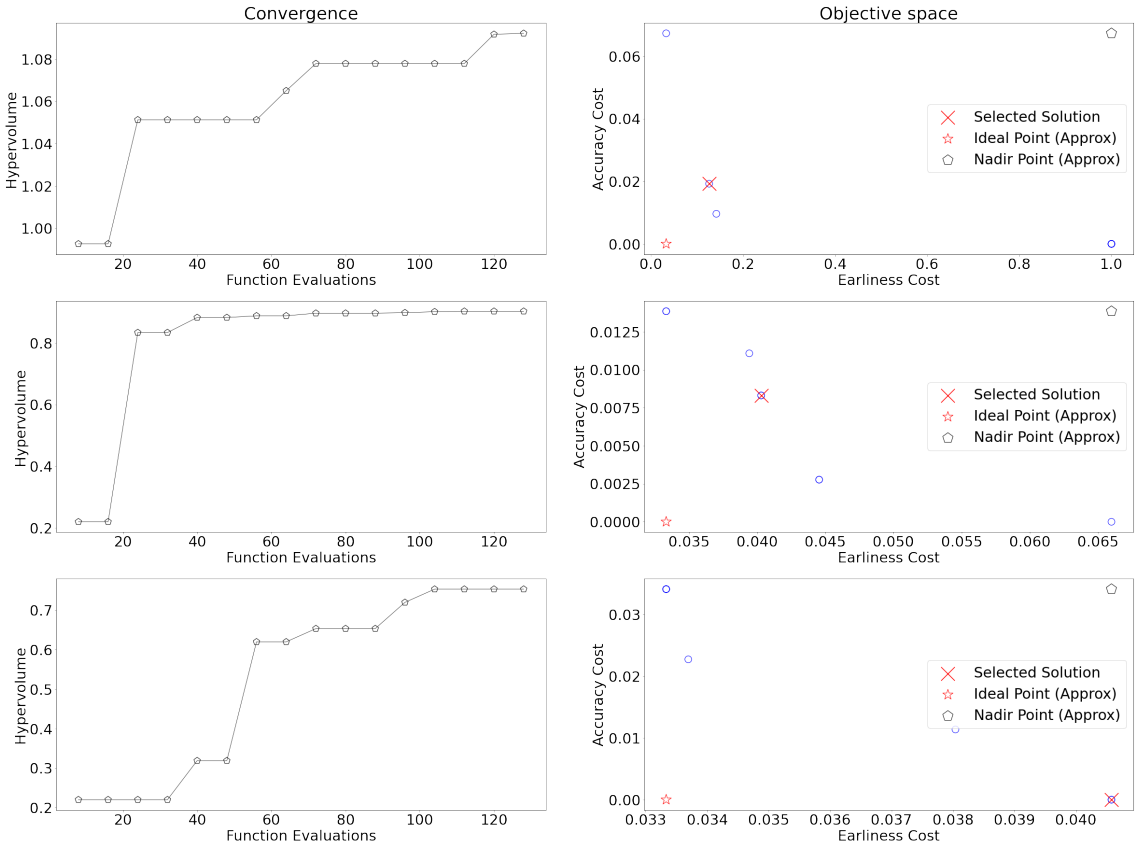


Figure 5.11: The figure shows results from the optimizations on the three different optimization data sets when using Tr^1. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost.

Table 5.2 shows the learned gamma parameters in the optimization of Tr^1 with regards to accuracy and earliness.

	Learned parameters		
Optimization set	γ_1	γ_2	γ_3
1	-0.04765224	0.77984266	-0.07144765
2	-0.09821839	0.82967275	-0.94366265
3	-0.07649895	0.71893477	-0.09503177

Table 5.2: The learned parameters by optimizing on three different data sets for Tr^1

5.2.1.4.2 Tr^2 The performance of the genetic algorithm for Tr^2 as well as the non dominated solution sets for the three data sets are presented in figure 5.12. Also

these optimizations converged more or less. Note that the genetic algorithm found an optimal solution from the start in the third case. Also note that the x-axis for the graphs showing the pareto fronts now show availability cost instead of earliness cost.

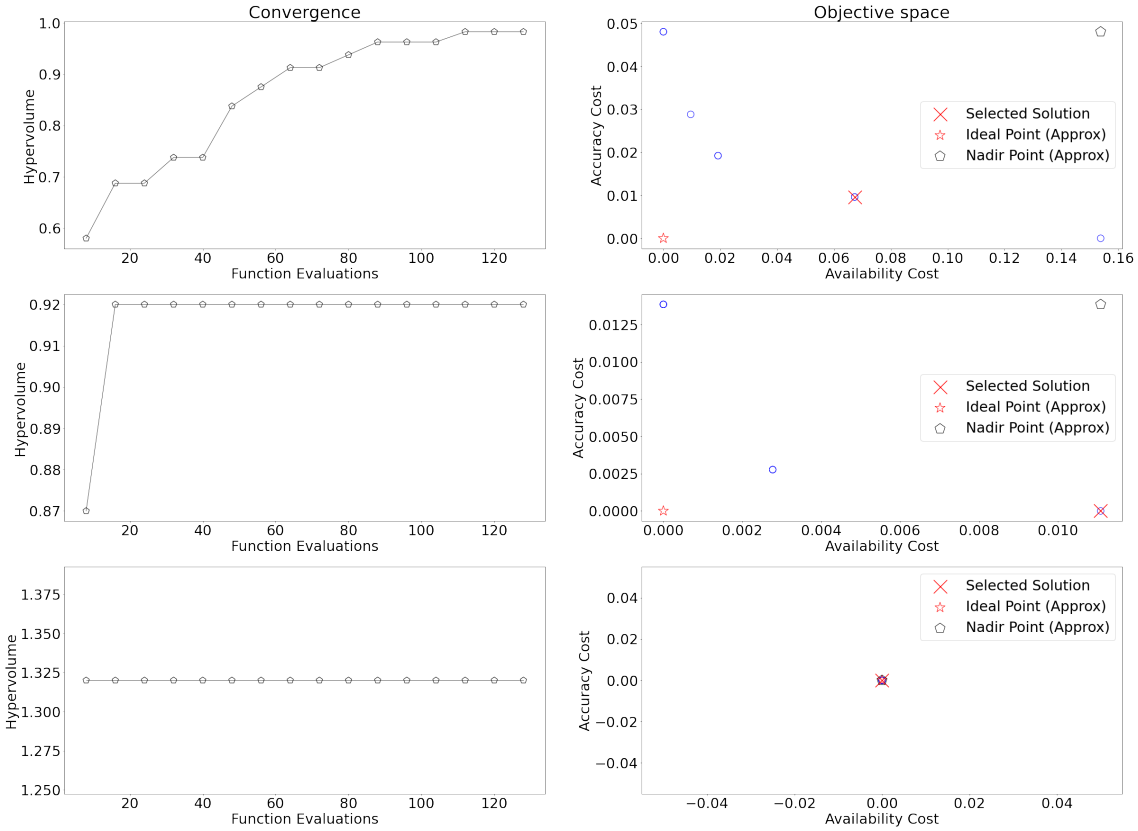


Figure 5.12: The figure shows results from the optimizations on the three different optimization data sets when using Tr^2. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost.

Table 5.3 shows the learned gamma parameters in the optimization of Tr^2 with regards to accuracy and availability. The learned parameters leads to a trigger function which favors a large difference between the highest and second highest probability to approve a prediction.

Optimization set	Learned parameters γ_1	γ_2
1	-0.55631805	0.66314203
2	-0.5910955	0.75623487
3	-0.41022409	0.99736572

Table 5.3: Table showing the learned gamma parameters from the optimization on the three different data sets for Tr^2.

5.2.1.4.3 Tr³ The performance of the genetic algorithm for Tr^3 as well as the non dominated solution sets for the three data sets are presented in figure 5.13. The first two optimizations converged more or less, but the last one may have needed a few more evaluations in the form of larger population size or number of generations.

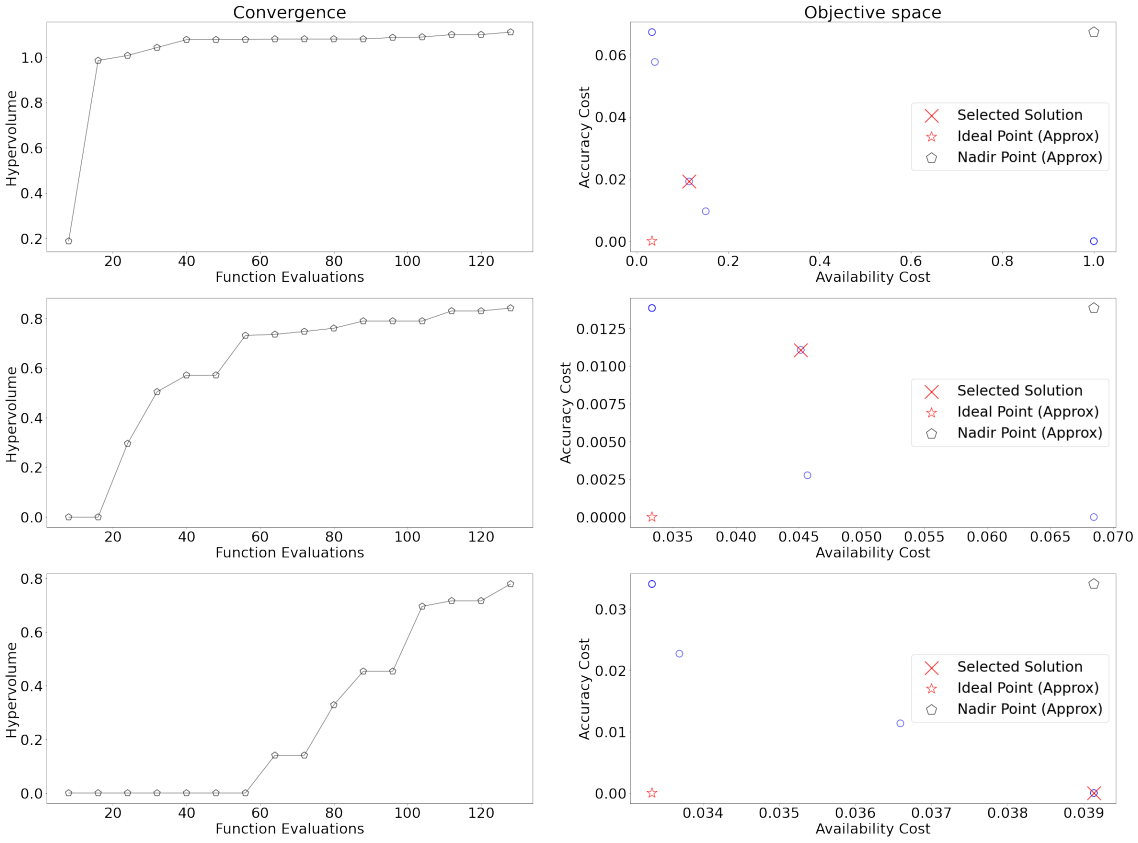


Figure 5.13: The figure shows results from the optimizations on the three different optimization data sets when using Tr^3. The left column shows the hypervolume. The right column shows the non dominated solution sets, with the addition of the nadir and ideal points, as well as a red X showing which solutions was chosen by the achievement scalarization function when using the weights (0.2, 0.8). The x-axis for the convergence-graphs on the left shows the number of function evaluations made by the algorithm. For the graphs on the right, the x-axis shows the earliness cost and the y-axis presents the accuracy cost.

Table 5.4 shows the learned gamma parameters in the optimization of Tr^3 with regards to accuracy and availability.

	Learned parameters			
Optimization set	γ_1	γ_2	γ_3	γ_4
1	-0.53672518	0.762149	0.1410882	0.33220885
2	-0.83180287	0.67649847	0.12448933	0.43641496
3	-0.04731121	0.58173865	0.22345755	-0.24882616

Table 5.4: Table showing the learned gamma parameters from the optimization on the three different data sets for Tr^3.

5.2.2 Testing of trigger functions

The trigger functions are compared as follows: Given that the trigger functions are optimized in three scenarios, with different training and optimization data sets as given by table 5.1, the testing will also be split in the three data sets, so that no information leakage is happening from training and optimization to testing. The testing will be done with the optimized parameters presented in tables 5.2,5.3 and 5.4, and the results from the three different test runs are aggregated.

The average accuracy, earliness and availability for each trigger function are presented in figure 5.14. The earliness, as presented here, is calculated as the average number of seconds passed before a prediction is made. Accuracy is the portion of correctly predicted sequences, and availability is shown in the legend and describes the fraction of sequences that got a prediction (within 30 seconds).

Accuracy and earliness are only calculated over the sequences where a prediction is made. Figure 5.14 shows that the Tr^2 achieved the highest accuracy, but also a somewhat higher time to prediction. It is worth noting that Tr^1 which is optimized on earliness and not availability still results in a perfect availability score.

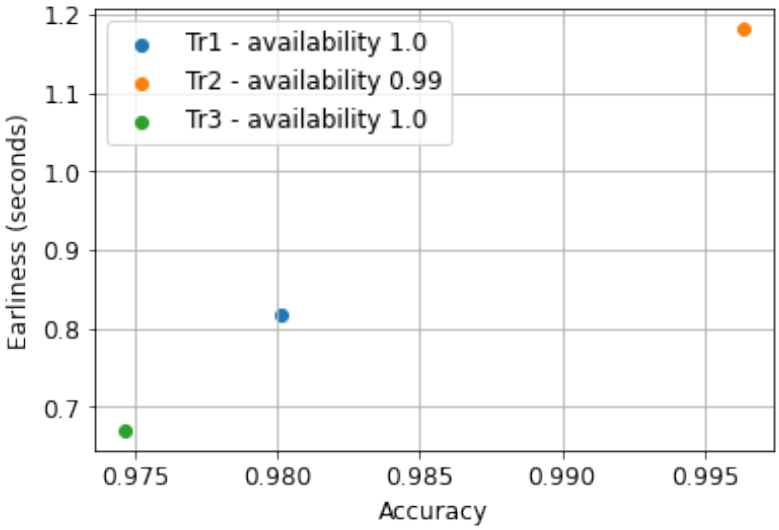


Figure 5.14: Scatter plot showing the performance of the three trigger functions with their optimized parameters on the testing data. The x-axis describes the fraction of correct predictions, the y-axis shows the average waiting time to prediction, in terms of the fraction of 30 seconds, and the availability of a prediction is presented in the legend together with the tested trigger function.

5.2.2.1 Highlighted cases

We investigate the driving scenarios for which the solution produces an incorrect classification, as well as the driving scenarios in which the solution is not immediately able to produce a classification. The trigger function used for all further result analysis is Tr^2, if nothing else is implied.

5.2.2.1.1 Incorrect classifications Two failed predictions were made by the solution using Tr^2. Figure 5.15 shows one failure case. It presents the Model Qualities up to the point of prediction. The left graph shows the MQs for the correct hypothesis, while the right graph shows the MQs for the selected, incorrect, hypothesis. Missing values are represented with the constant 1 to make the graphs more readable. Note the lack of one of the Model Qualities types (R1 Type) for the correct hypothesis for most of the sequence up until prediction. Figure 5.16 shows the averaged MQs up until each time step, which is what is used as input to the probabilistic classifier.

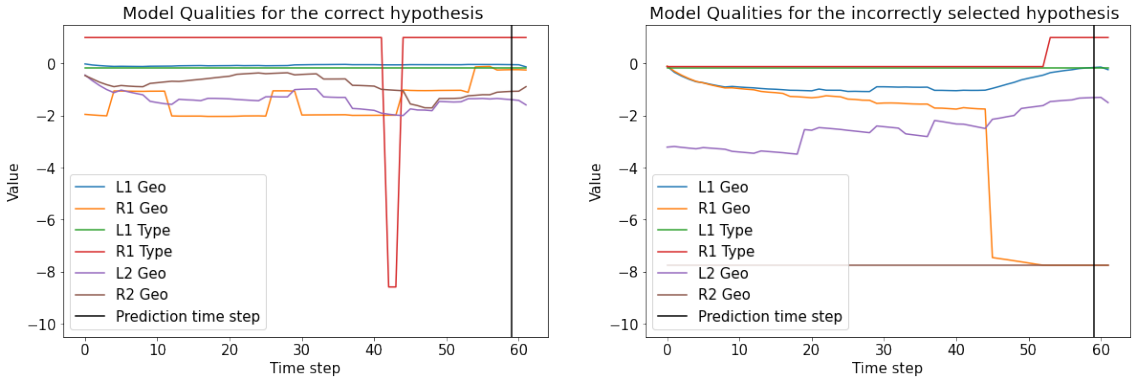


Figure 5.15: Model Qualities for the correct hypothesis to the left and the incorrect but selected hypothesis to the right. The y-axes shows the MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected.

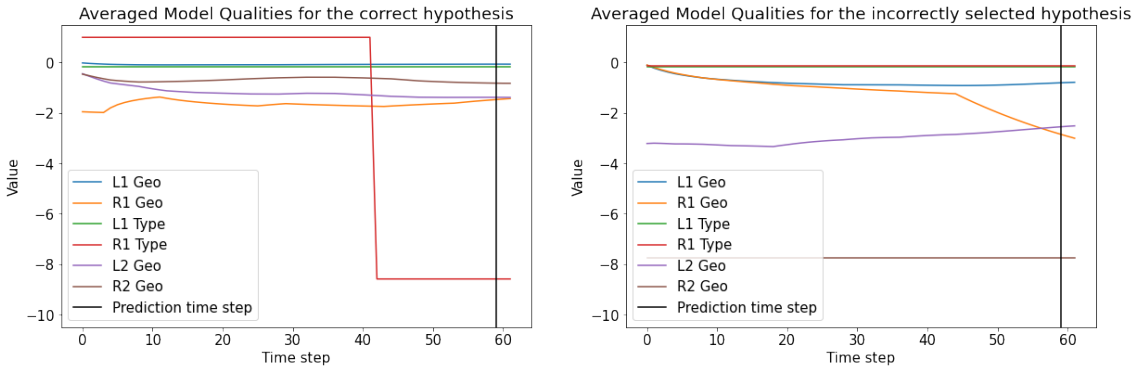


Figure 5.16: Averaged Model Qualities for the correct hypothesis to the left and the incorrect but selected hypothesis to the right. The y-axes shows the averaged MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected.

It is worth noting that the correct hypothesis produces a short amount of bad R1 Type qualities after around 40 time steps, figure 5.15. Given the strategy of using the current average quality as input to the model, this results in bad R1 Type qualities for every time step after this, which can be seen in figure 5.16.

5.2.2.1.2 Late prediction In 70 out of 553 sequences in the test sets, the trigger function did not trigger the first time, and a prediction was made later in the sequences. For these late predictions, the average time to prediction was 8.1 seconds. One example of a late prediction is shown in figures 5.17 and 5.18, where figure 5.17 shows the momentary MQs for the two hypotheses with the highest probability from the probabilistic classifier, with the left column being the correctly selected hypothesis and the right column showing the second most probable hypothesis, as given by the probabilistic classifier. Figure 5.18 shows the averaged MQs up until each time step for the same two hypotheses. The second most probable hypothesis

have good MQ type values up until time step 130. A decline in R1 type (red line) is clearly seen in figure 5.18 up until a correct prediction was made at time step 199. The probabilities p' and p", the highest and second highest probabilities outputed by the probabilistic classifier, for the time steps up until prediction are shown in figure 5.19. It clearly shows that the belief in the second most probable hypothesis, as given by the probabilistic classifier, decreases during the last 80 time steps before the trigger function triggers a prediction to be made.

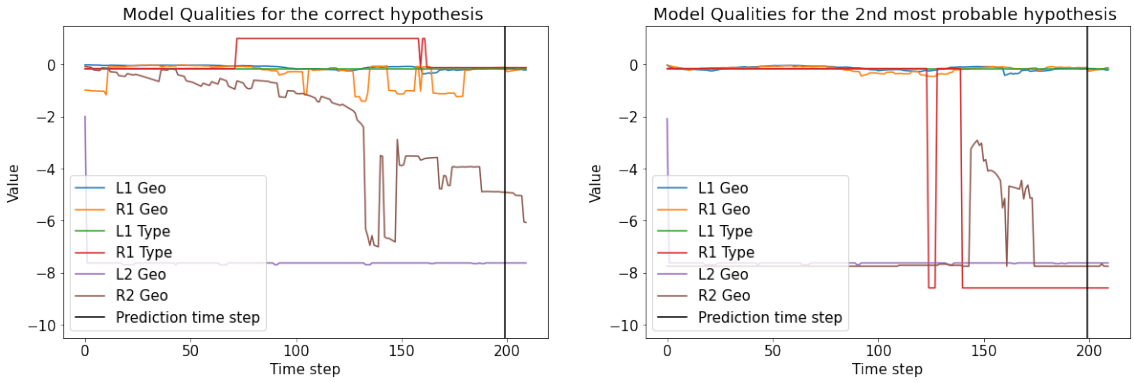


Figure 5.17: Model Qualities for the correct (and highest probable) hypothesis to the left and the second most probable hypothesis to the right. The y-axes shows the MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected.

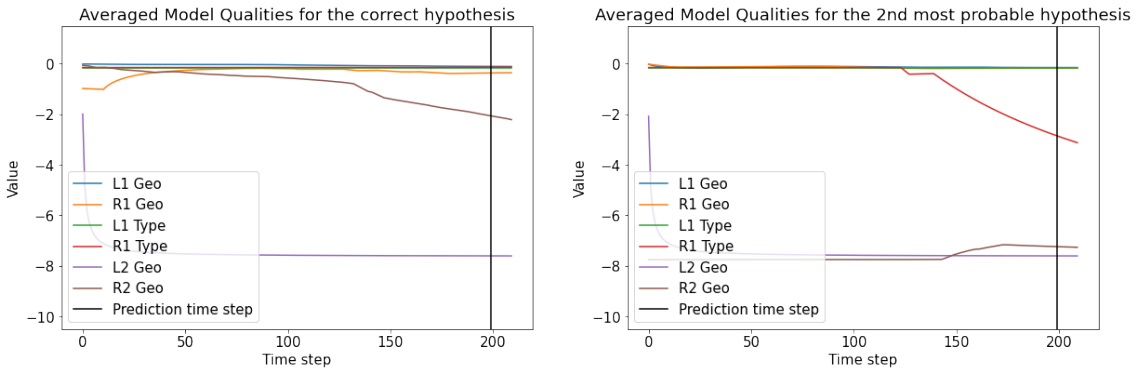


Figure 5.18: Averaged Model Qualities for the correct (and highest probable) hypothesis to the left and the second most probable hypothesis to the right. The y-axes shows the averaged MQ values and the x-axis represent the time steps. The black vertical line represents the time step at which the wrong hypothesis was selected.

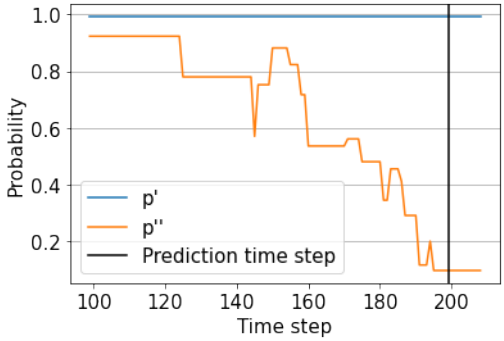


Figure 5.19: The two highest probabilities from the probabilistic classifier, p' and p" respectively, for the time steps before prediction. The black vertical line represents the time step at which a prediction was triggered.

5.2.2.1.3 No prediction 5 out of 553 sequences did not get a classification, i.e. the trigger function never triggered in these sequences. Figure 5.20 shows the top two probabilities, p' and p", in one of these cases. The blue line represent the probabilities for the correct hypothesis while the orange line represent the probabilities for an incorrect hypothesis.

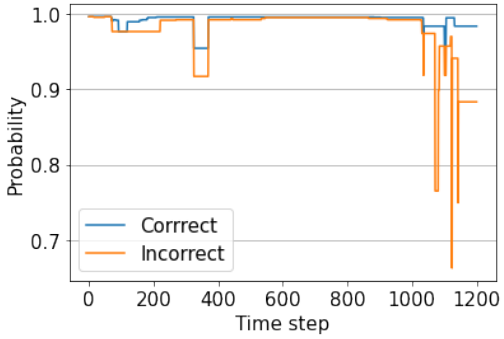


Figure 5.20: The two highest probabilities from the probabilistic classifier, p' and p" respectively, for a sequence in which the trigger function never triggered a prediction.

Figure 5.21 shows the top two probabilities for another sequence in which no prediction was made. Note that no prediction is triggered even if the probabilistic classifier is confident in the correct hypothesis and the difference between the correct and incorrect hypothesis is more than 0.6 in the end.

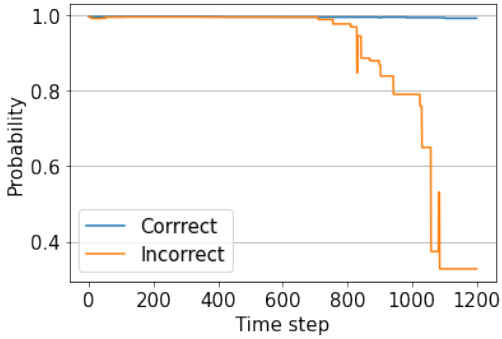


Figure 5.21: The two highest probabilities from the probabilistic classifier, p' and p" respectively, for a sequence in which the trigger function never triggered a prediction.

In figure 5.21 we see a case in which no prediction is made for the entire sequence. The belief in hypothesis 1 starts to decrease from around 700 time steps in, but no prediction is made for the remaining 500 time steps even though the probability of 1 being true drops to below 0.4. For this sequence, the type MQs starts to decrease at around time step 700.

This is a consequence of the heavy preference that was put on accuracy over availability. Using a moving window average instead of including the full time series prefix would mean that the features would be more reactive to a change in the MQs. This would in turn lead to a quicker and more severe change in the probabilities, and a prediction may have been possible before the end of the sequence.

6

Discussion and Conclusion

This chapter contains a discussion of the results and the project as a whole in section 6.1, and our main conclusions are summarised in section 6.3. A number of suggestions for future work are also provided.

6.1 Results discussion

As a solution to the hypothesis-inference localization problem, a two-tiered approach including a supervised machine learning classifier and an optimized trigger function has been shown to perform well. The results from the two parts of the hypothesis-inference solution will be discussed in the following sections.

6.1.1 Data

One main finding is that the data that was used is limiting. One reason for this is that many of the sequences cannot be annotated using our procedure of automated annotation which means that they are discarded. This is problematic as it cannot be assumed that this removal will not shift the distribution of the data set.

For example, lane merges may lead to ambiguity in annotation as the vehicles get close to each other, and the entire sequence may end up being discarded. In this case, the lane merge scenario type may be entirely removed, or at least underrepresented, in the data set.

The requirement for sequences to be fully annotated was put in place to ensure that only sequences with a clear answer as to which hypothesis is true was used. As we have seen, this may lead to imbalance of the data set or removal of problematic sequences which we would expect an online solution to able to handle.

The data set also seems to contain insufficient features for determining the true hypothesis with certitude in certain scenarios. As lane geometries tend to be similar due to the parallel lanes, there is a reliance on lane types. This leads to an inability to predict the correct hypothesis in cases with similar type data.

The data set that is being used has been processed in several steps and is reliant on the correctness of the map, the perception system for the lane edges, the algorithm

for computing the MQs, and the reference model. Errors can be introduced in any of these steps.

6.1.2 Probabilistic classifier

Several different machine learning classifiers have been compared, such as Naive Bayes, Logistic Regression and Random Forest classifiers. It was found that they all performed similarly well in the binary classification task, deciding if a sample belonged to a true or false hypothesis.

The performance did not seem to differ significantly between the same model trained and tested on differently sized rolling window averages. The similarity in performance may be due to the fact that the data is low-dimensional, and does not contain any complex synergies that a more advanced classifier would be able to capture. This is further supported by the "feature importances" extracted from the probabilistic classifier which clearly shows that the lane types are the strongest indicators. It is also confirmed in the constructed feature set as presented in section 5.6, where the most relevant factor in the classification is the combination of the lane types. A classifier using this simple constructed feature set manages to achieve a performance similar to that of the state-of-the-art CIF-classifier.

However, since the CIF creates many different features from the input data, it requires some strategy for handling the missing data. Imputing with a constant (as in the case of the decision tree ensembles) or other methods will affect the different calculated measures. A CIF would likely work better if used with more dense input data, as opposed to the Model Quality data set, which by nature contains features that are sparse, such as the secondary lane markers.

6.1.3 Trigger function

The results from the comparison of the different trigger functions are similar, and the performance is mostly good. We suspect that this is due to the limitations in the data. The original data set contained many driving sequences that had to be removed due to being short, and the automated annotation procedure did in many cases not succeed at annotating full sequences. We elected to only keep the sequences that were of full length and contained annotations for the full sequence. This may have led to a heavily refined data set that contains driving sequences that are easily classified. On the other hand, this choice was motivated by the bad quality data which was found in sequences lacking some of the labelling.

Although the convergence of the optimization of all different trigger function parameters cannot be guaranteed, and some of the optimization runs may have done well with a few more function evaluations, we think that the majority of optimization runs would not create better solution sets by letting the genetic algorithm run for longer. Tr^2, the simple trigger function optimized on availability and accuracy seems to give the best accuracy while still keeping the availability high and earliness quite low. An interesting thing to note is that Tr^1, which was not optimized on

availability, still gets a full availability in the test. Overall all three trigger functions performed quite well and it is difficult to say with certainty that one is much better than another.

6.1.4 Hypothesis-inference

When evaluating specific cases in the hypothesis-inference with Tr^2, some cases exist where the system makes a correct prediction by waiting for a better information state before making a decision. Finding these cases in which an insufficient information state is identified, a rejection is performed, and later a correct classification is made, tells us that the application of these techniques for solving the lane-level localization problem is a sound approach.

Some sequences were predicted very late (or not at all) even though the information state seemed to improve and be sufficient for classification. For this project, we elected to heavily prioritize accuracy over earliness, which leads to this inertia in the model. For a real implementation of this architecture for real-time localization, this will need to be adapted.

One method is to use a windowed average instead of an average over the entire time-series prefix to improve reactivity. This only become more important as sequences get longer than the 30 second sequences that were used for this study. In essence, the choice of how the input data to the model has been smoothed over time will probably be dependent on a trade off between accuracy and reactivity, where less smoothed data will lead to a model that will be able to be more reactive to changes, but at the same time may be too nervous in its prediction, risking a higher false positive rate and therefore a lower accuracy. On the other hand, in scenarios where the MQs in the initial part of a sequence are equally good for several hypotheses and therefore not enough for the probabilistic classifier to differentiate between a correct and incorrect hypothesis, a more reactive probabilistic classifier may be needed to quickly identify the correct from the incorrect hypothesis when their MQs start to diverge.

This study has been made on quite strongly filtered data. In this scenario the use of early time series classification techniques seem to work well for the localization of autonomous vehicles. The choice of probabilistic classifier and trigger function does not seem to make a big difference to the performance of the model. A natural next step to this study would be to investigate how this method of hypothesis inference would perform on more up-to-date, less filtered and more general data. Furthermore, since this classification procedure is highly dependent on lane type data, the incorporation of other meaningful features would be something to consider to increase the robustness of the probabilistic classifier.

A robust model can handle a wide set of driving scenarios. If the training data is over-represented by straight highway driving, it may end up performing well in these scenarios, but less well in more uncommon scenarios, such as in the presence of ramps or intersections. This tells us that it is important to consider the balance of the data

set. This may be done by categorizing driving scenarios and using a balanced and varied set of sequences. Examples of categories can include the presence of on- and off-ramps, lane merges, number of lanes, etc.

6.2 Limitations and scope

Given the relatively short time frame of 20 weeks for the completion of the project, it has been approached with a proof-of-concept mindset, where simplifications have been made in several parts of the research, development and testing to be able to reach a complete working solution:

- Machine learning classifiers have been trained with default parameters.
- The data set that is being used was collected between September of 2020 and May of 2021.
- Only fully annotated sequences are used.
- No deep learning approaches has been evaluated since we believe that working with deep learning would have taken a lot of time that could be directed elsewhere.

6.3 Conclusion

We conclude that early time series classification techniques are promising for the purpose of lane-level localization of autonomous vehicles. A well performing solution was found using a Gradient Boosting classifier together with a trigger function with parameters optimized on accuracy and availability. This solution achieved an accuracy of 99,5% as well as an availability of prediction of 99%. The solution is trained and tested on filtered data which means that real world performance may differ somewhat.

Different shapes of the linear inequality trigger, as well as the probabilistic classifier, show similar results which tells us that the data is the limiting factor. Our conclusion is that the low dimensionality of the data and the heavy influence of lane marker types means that more complex models show no benefit over simpler models. The majority of driving scenarios are easily classified, but for some types of scenarios we have identified that our proposed solution is insufficient. Due to the reliance on lane type data, these scenarios are mostly related to missing or similar lane type data.

Bibliography

Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3).

Cui, D., Xue, J., and Zheng, N. (2016). Real-Time Global Localization of Robotic Cars in Lane Level via Lane Marking Detection and Shape Registration. *IEEE Transactions on Intelligent Transportation Systems*, 17(4).

Deb, K. and Deb, K. (2014). Multi-objective Optimization. In *Search Methodologies*, pages 403–449. Springer US, Boston, MA.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dempster, A., Petitjean, F., and Webb, G. I. (2019). ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels.

Faouzi, J. and Janati, H. (2020). pyts: A python package for time series classification. *Journal of Machine Learning Research*, 21(46):1–6.

Fulcher, B. D. and Jones, N. S. (2017). hctsa: A Computational Framework for Automated Time-Series Phenotyping Using Massive Feature Extraction. *Cell Systems*, 5(5).

Ghalwash, M. F., Ramljak, D., and Obradović, Z. (2012). Early classification of multivariate time series using a hybrid HMM/SVM model. In *Proceedings - 2012 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2012*.

Hatami, N. and Chira, C. (2013). Classifiers with a reject option for early time-series classification. In *Proceedings of the 2013 IEEE Symposium on Computational Intelligence and Ensemble Learning, CIEL 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*.

Huang, M. X. and Li, Y. (2021). Tapnet: The design, training, implementation, and applications of a multi-task learning cnn for off-screen mobile input. In *Conference on Human Factors in Computing Systems - Proceedings*.

Jensfelt, P. and Kristensen, S. (2001). Active global localization for a mobile robot

using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760.

Jo, K., Jo, Y., Suhr, J. K., Jung, H. G., Sunwoo, M., Li, F., Bonnifait, P., Ibanez-guzman, J., Ibañez-guzmán, J., Li, F., Bonnifait, P., Using, J. I.-g., Definition, H., Gnss, E., Li, F., Bonnifait, P., Ibanez-guzman, J., Cardenas, I. L., Du, J., and Barth, M. J. (2017). Using High Definition Maps to Estimate GNSS Positioning Uncertainty To cite this version : HAL Id : hal-01569092 Using High Definition Maps to Estimate GNSS Positioning Uncertainty. *IEEE Transactions on Intelligent Transportation Systems*, 9(6).

Joubert, N., Reid, T. G. R., and Noble, F. (2020). Developments in Modern GNSS and Its Impact on Autonomous Vehicle Architectures. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 2029–2036. IEEE.

Laconte, J., Kasmi, A., Aufrère, R., Vaidis, M., and Chapuis, R. (2022). A survey of localization methods for autonomous vehicles in highway scenarios.

Lauxmann, R. (2015). Sense-Plan-Act – the role of chassis systems.

Lee, S., Choi, J., and Seo, S. W. (2021). Ego-lane index-aware vehicular localisation using the DeepRoad Network for urban environments. *IET Intelligent Transport Systems*, 15(3).

Li, F., Bonnifait, P., and Ibanez-Guzman, J. (2018). Estimating localization uncertainty using multi-hypothesis map-matching on high-definition road maps. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-March.

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., and Jones, N. S. (2019). catch22: CAnonical Time-series CHaracteristics.

Marler, R. and Arora, J. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395.

Middlehurst, M., Large, J., and Bagnall, A. (2020). The Canonical Interval Forest (CIF) Classifier for Time Series Classification. In *Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020*.

Mori, U., Mendiburu, A., Miranda, I. M., and Lozano, J. A. (2019). Early classification of time series using multi-objective optimization techniques. *Information Sciences*, 492.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., and Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2).

Santos, T. and Kern, R. (2017). A literature survey of early time series classification and deep learning. In *CEUR Workshop Proceedings*, volume 1793.

Schäfer, P. and Leser, U. (2020). TEASER: early and accurate time series classification. *Data Mining and Knowledge Discovery*, 34(5).

Wang, S. and Jiang, J. (2015). Learning Natural Language Inference with LSTM.

Wang, Z., Yan, W., and Oates, T. (2016). Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline.

Xing, Z., Pei, J., and Yu, P. S. (2012). Early classification on time series. *Knowledge and Information Systems*, 31(1):105–127.

Yao, L. and Guan, Y. (2018). An Improved LSTM Structure for Natural Language Processing. In *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, pages 565–569. IEEE.