CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Autoencoder and Active Learning to Reduce False Positive Warnings in a Slippery Road Alert System

Using unlabeled multivariate time series

Master's thesis in Computer science and engineering

Philip Axenhamn

Andreas Greppe

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Autoencoder and Active Learning to Reduce False Positive Warnings in a Slippery Road Alert System

Using unlabeled multivariate time series

Philip Axenhamn

Andreas Greppe

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Autoencoder and Active Learning to Reduce False Positive Warnings in a Slippery Road Alert System
Using unlabeled multivariate time series
Philip Axenhamn
Andreas Greppe

Supervisor: Selpi, Department of Computer Science and Engineering
Advisor: Per-Olof Nilsson, Volvo Cars
Examiner: Peter Damaschke, Department of Computer Science and Engineering

Master's Thesis 2023
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
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Abstract

Modern vehicles are commonly equipped with a slippery road alert (SRA) system that warns drivers of slippery roads. Current implementations of this system occasionally produce warnings when the road is not slippery. These warnings are called false positives and can harm the system's trustworthiness. In this thesis, we propose a false positive filter capable of reducing the number of false positive alerts generated by an SRA system based on two machine learning techniques: autoencoder and active learning. The available data was completely unlabeled and contained few informative features. The analysis of this data showed that vehicles send bursts of data points forming sequences. Due to the limitations of the data, multi-variate time series were constructed with the idea that the sequential data might reveal more about the situation than a single-point measurement. Furthermore, the sequences were grouped into true- and false-positive classes based on assumptions of the causes for the alerts, such as driving on ice or a speed bump. The sequential data was used to train GRU- and LSTM-based autoencoders and classifiers to detect sequences that correspond to false positive situations such that they can be removed. The hyperparameters for the models were tuned using Optuna and the best-performing models with the most optimal hyperparameters were further evaluated. Since the data was not labeled, the actual performance of the proposed solution could not be assessed. Instead, the evaluation was based on computing the proportion of remaining assumed true positive (ATP) sequences and assumed false positive (AFP) sequences after filtering. The results show that the LSTM autoencoder could find patterns in the sequential data and was able to remove 43% of the AFP sequences while retaining 90% of the ATP sequences. The active learning approach proved to not work well with the available data.

Keywords: slippery road alert, false positive, machine learning, unsupervised learning, semi-supervised learning, autoencoder, active learning, anomaly detection

Acknowledgements

We would like to express our gratitude to everyone who has supported us during the project. We want to thank our supervisor Selpi from Chalmers, who gave us valuable suggestions and helped us form ideas. We also thank the people at Volvo Cars who made the project possible. A special thanks to Fredrik and Per-Olof who provided feedback and help throughout the project.

Philip Axenhamn & Andreas Greppe
Gothenburg, 2023-06-19

Contents

List of Figures

List of Tables

List of Acronyms

ABS Anti-lock Braking System.

ADAS Advanced Driver Assistance System.

AE Autoencoder.

AFP Assumed False Positive.

AL Active Learning.

ANN Artificial Neural Network.

ATP Assumed True Positive.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

GRU Gated Recurrent Unit.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

ML Machine Learning.

MSE Mean Squared Error.

RNN Recurrent Neural Network.

SRA Slippery Road Alert.

TCS Traction Control System.

1

Introduction

Slippery road surfaces are a common cause of car crashes in northern countries. According to Trafa (Swedish government agency for traffic analysis), 33% of all road-traffic accidents reported during 2022 in Sweden occurred on wet or icy road conditions [1]. To increase vehicle and traffic safety, modern vehicles are equipped with Advanced Driver Assistance System (ADAS) that aims to assist drivers in different driving situations, including difficult road conditions. A subsystem of ADAS is the slippery road alert (SRA) system that warns drivers of nearby slippery roads via a cooperative intelligent transport system (C-ITS) where data, including information about road slipperiness, is anonymously shared with nearby vehicles via a cloud-based solution. These warnings make the driver aware of a potentially slippery road ahead such that the vehicle can be manoeuvred to avoid slipping. Inaccurate estimations of the road slipperiness in this system can lead to either misses of slippery roads (false negatives) or false warnings (false positives). Hence, generating accurate slippery road alerts is significantly important for vehicle and traffic safety and the reliability of the system.

1.1 Problem Formulation

In an SRA system, vehicles share friction data to the cloud where it is processed to generate SRAs that are sent to other nearby vehicles. Current implementations of SRA systems rely on the aggregation of estimated friction coefficients between tire and road surface based on measurements from in-vehicle sensors. In [2], the authors proposed a sensor data fusion approach for friction estimation based on the brush model and Kalman filter. Furthermore, a more accurate friction estimation can be made using systems such as the anti-lock braking system (ABS) and the traction control system (TCS). However, current implementations of SRA systems occasionally produce false positive alerts. Hence, drivers receive warnings of slippery roads when the road is not slippery. Reduction of false positive SRAs is important since it would increase the reliability and trustworthiness of the system which in turn can lead to drivers reducing the speed of the vehicle instead of ignoring the warnings.

1.2 Aim and Objectives

This thesis aims to reduce the number of false positive alerts generated by an SRA system based on two machine learning techniques: autoencoder and active learning. The proposed solution will be a false positive filter intended to remove sequences that correspond with false positive situations e.g. driving on a speed bump. The primary objectives of this thesis can be divided into two parts: First, the causes of potential false positive alerts will be investigated by analyzing the available data. This is to better understand in which situations the assumed false positive alerts commonly occur to motivate decisions taken in the study. Second, we aim to improve an SRA system by focusing on reducing the number of false positive alerts through the use of the previously mentioned machine learning approaches.

1.3 Contributions

Numerous studies have investigated techniques to estimate the friction between tire and road surface. However, reducing false positive SRAs by the use of machine learning models solely on unlabeled and estimated friction data in combination with the weather- and road data has never been done before this study. In this thesis, we use two different machine learning approaches based on autoencoder and active learning to reduce the number of false positive SRAs. Both of the approaches will be evaluated using unlabeled data only. Furthermore, the study provides an analysis of potential causes of false positive alerts in an SRA system based on the available data. Finally, an approach for creating sequential data from spatial-temporal data points based on time series clustering and DBSCAN is proposed.

1.4 Limitations

The limitations of this study were mostly related to the available data and the evaluation of the proposed solution.

- **Spatial- and temporal limitations in data:** A data set was not available at hand and was therefore needed to be constructed during the project. The data sets used for training the machine learning models were gathered within a limited time frame, more specifically during winter and early spring. Hence, the generated data sets lack data from both summer and autumn where false positive alerts probably are easier to distinguish from true positive alerts. Additionally, the data was gathered from vehicles in Gothenburg, Sweden.
- **Feature limitations in data:** The most important available features in the available data were the estimated friction coefficient and friction quality. Properties of the vehicle such as its velocity, weight, and lateral- and longitudinal forces on the wheels, were not available but would probably be beneficial for detecting false positive alerts.
- **Unlabeled data:** The available friction data was unlabeled meaning that ground

truths of true- and false positives were not available. Creating a labeled data set would be a time-consuming process and due to limited time for the project, this was avoided. Hence, unsupervised and semi-supervised methods were used and some of the choices made in this study were based on assumptions motivated by the literature study and data analysis.

- **Evaluation limitations:** Evaluation of unsupervised machine learning models is generally challenging since there is no ground truth to compare with. Usually, at least a small set of labeled data is collected to have something to evaluate against. However, a labeled data set could not be created due to practical difficulties and time constraints. Hence, determining if the proposed solution reduces false positive alerts was not manageable. Instead, the evaluation of the proposed solution was mostly based on comparing the machine learning models on assumed true- and false-positive sequences using metrics that do not rely on ground truths.

1.5 Thesis Outline

The rest of this paper is structured as follows:

- Chapter 2 presents previous work and research related to this study.
- Chapter 3 provides a theoretical background of the most important concepts and techniques used in this project.
- Chapter 4 provides a description and analysis of the available data.
- Chapter 5 describes the methods used and how the project was conducted including data processing, choice of architecture, development of the machine learning models, and finally the evaluation of the proposed solution.
- Chapter 6 presents the results of the experiments and final evaluation of the autoencoder and active learning approach.
- Chapter 7 discusses the results and concludes the thesis with a discussion of ethical- and legal considerations, insights on the topic, and future work.

2

Related Work

False positive alerts can be considered to be anomalies and there are numerous studies made on how to detect anomalies in time series. In [3], the authors discussed common complexities and challenges in the field of anomaly detection using deep learning techniques. First, some anomalies might be unknown until they occur which makes them difficult to prevent. Second, there might be heterogeneous anomaly classes i.e. anomalies with different root causes and abnormal characteristics. Third, anomalies are typically rare and only occur on special occasions which can lead to class imbalance.

In [4], the authors pointed out that anomaly detection is different from traditional machine learning where the goal is to find similarities rather than dissimilarities in the data. Furthermore, the authors mentioned that the evaluation of anomaly detection systems is crucial but difficult to perform and even claimed that it is even more challenging than developing the system itself. One of the most challenging aspects of evaluating anomaly detection systems is the lack of a labeled data set. Furthermore, in other fields of machine learning, there are often public data sets available that make the results directly comparable with other methods used in previous studies. In contrast, researchers in the field of anomaly detection are forced to create their own data sets which means that the results are not directly comparable with results from previous studies.

For the reasons previously mentioned, anomaly detection is typically defined as an unsupervised learning task. In [5], k-means clustering was used to reduce anomalies in the field of computer networks. Using this approach, it was possible to achieve an anomaly detection rate of 82.92%. The use of clustering-based methods is typically less complex than machine learning models and can therefore be easier to implement. However, other methods might be more suitable than clustering depending on the available data.

Autoencoders are commonly used as an unsupervised method for anomaly detection and can be applied to many different types of complex data including images and time series. In [6], autoencoders were applied to reduce false positives in a handgun detection system. The results showed that the autoencoder could reduce the number of false positives of detected guns by 78%.

In the work [7], the authors trained LSTM-based autoencoders on time series for anomaly detection. The LSTM autoencoders achieved an accuracy rate of 87% in

detecting anomalies.

A more recent approach for anomaly detection in time series involves the utilization of transformers, as proposed in [8]. The experimental results revealed an improvement in F1-score by up to 17% and a remarkable 99% decrease in training time compared to the baseline models. While transformer-based autoencoders share similarities with traditional RNN-based autoencoders, they offer advantages such as enhanced performance and execution time. However, it is important to note that these benefits come at the expense of a more intricate architecture, which may pose implementation challenges.

Moreover, research efforts have also been directed toward the prediction of road states using various machine learning techniques. Multiple studies have explored approaches, utilizing data from external sensors such as GPS, accelerometers, and gyroscopes attached to vehicles. In [9], the authors employed unsupervised methods, including self-organizing maps (SOM) and k-means clustering in combination with the aforementioned sensor data to group roads based on their roughness and condition. Conversely, in [10] the authors implemented a supervised learning approach to predict road obstacles using the same type of sensor data. These studies show the potential for using sensor data to directly estimate and identify road obstacles and conditions from data available in the vehicles.

3

Theory

This chapter provides a theoretical framework of the key concepts of the project. Section 3.1 describes the fundamentals of friction and how it occurs between tire and road surface. Section 3.2 describes the most critical factors affecting the estimated friction coefficient. Section 3.3 describes important concepts within machine learning relevant to the study. Section 3.4 describes the DBSAN clustering algorithm used in the sequence generation process. Section 3.5 provides a theoretical background of artificial neural network, recurrent neural network, LSTM, GRU, and autoencoder. Finally, Section 3.6, describes spatial hashing which is a method that was used in the creation of the data sets.

3.1 Friction Between Tire and Road Surface

Friction is the force that counteracts the relative motion between two surfaces that are in contact with each other. The coefficient of friction (COF), often written as μ, is defined as the ratio between the impressed force and the normal force of an object and ranges from 0 (no friction) to 1 (full friction). For a moving object e.g. a rotating wheel on a flat horizontal surface, the coefficient of friction is defined as the quotient between the horizontal friction force (commonly called rolling resistance) F and the vertical normal force F_N [11].

$$\mu = \frac{F}{F_N} \tag{3.1}$$

As shown in Figure 3.1, the friction force F from the ground acts in the opposite direction to the direction of motion of the wheel at the point of contact. On the vertical axis of the wheel, the normal force F_N that is perpendicular to the road surface is the force the ground is acting on the wheel. Conversely, according to Newton's third law of motion, the wheel is acting with an equal amount of force F_W on the ground which is calculated by multiplying the mass with the acceleration due to gravity [11]. The friction force F between tire and road surface enables the vehicle to move forward and is essential for the maneuvering of the vehicle in both the lateral- and longitudinal directions. In general, higher friction means more control of the vehicle. In cases when the coefficient of friction is low, the vehicle starts to slip and the driver loses control of the vehicle. Hence, accurate estimation

of the friction between tire and road surface is important for road manufacturers to improve road and traffic safety.

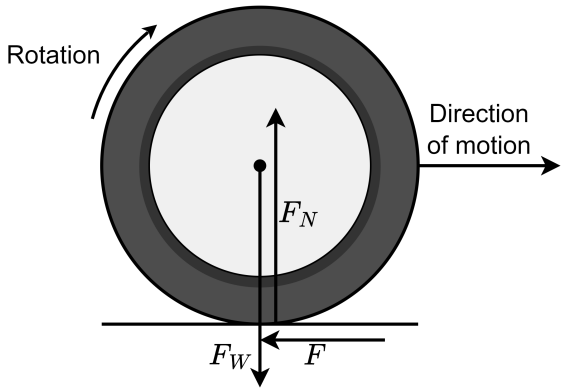

Figure 3.1: Simplified visualization of the forces acting on a rotating wheel.

3.2 Estimating The Coefficient of Friction

The coefficient of friction (COF) can be estimated for each wheel or as the average tire-road friction coefficient for all wheels of the vehicle. In the case of the latter, the coefficient of friction is assumed to be equal for all wheels. Conversely, for individual tire-road friction coefficient estimation, this assumption is not made [12]. In this thesis, the coefficient of friction between tire and road surface was estimated as the average friction for all wheels. Average friction coefficient estimation is accomplished by using slip-slope methods, extended Kalman filter, or lateral-dynamics-based methods [12].

Slip-slope methods utilize the relationship between the normalized longitudinal forces acting on the wheels and the relative motion between tire and road surface, referred to as slip [12] to estimate the COF between tire and road surface. Tire slip has been proven to have a high correlation with the COF and can be described as the amount of wheel lock where 0% slip corresponds to a free-rolling wheel and 100% corresponds to a fully-locked wheel [13]. The coefficient of friction increases rapidly until the peak friction is reached which typically occurs when slip is 10-20%. Then, the coefficient of friction decreases slowly up to 100% slip, i.e. when the wheel stops rotating. To avoid sliding when braking on icy roads the relationship between the coefficient of friction and slip is used in the anti-lock brake system (ABS) to determine when it needs to activate [13].

The interaction between tire and road surface is a complex event that is affected by a variety of parameters including vehicle weight, slip, road condition, and surface texture. Additionally, surrounding environmental factors such as ice, snow, slush, water, and gravel have an impact on the friction that occurs between tire and road surface. Some of the most highly influential factors are divided into four categories presented in Table 3.1, modified from [13].

Table 3.1: Factors affecting the estimated friction between tire and road surface.

Vehicle operating parameters	**Tire properties**	**Road characteristics**	**Environmental factors**

• Longitudinal motions
  ◦ Vehicle speed
  ◦ Braking action
• Lateral motions
  ◦ Turning
  ◦ Overtaking
	• Tread design and condition
	• Inflation pressure
	• Rubber composition and stiffness
		• Road type
		  ◦ Asphalt
		  ◦ Brick
		  ◦ Gravel
		• Unevenness
		  ◦ Pothole
		  ◦ Speed bumps
			• Weather
			  ◦ Temperature
			  ◦ Precipitation
			  ◦ Humidity
			  ◦ Wind speed
			• Contaminants
			  ◦ Anti-skid materials (salt, sand)
			  ◦ Dirt, mud, oil spill

3.2.1 Vehicle Operating Parameters

The estimated friction coefficient between tire and road surface depends on how the vehicle is maneuvered. As previously mentioned, the friction coefficient is estimated based on the forces acting on the wheels. Since driving maneuvers such as accelerating, braking, and turning affects these forces, the estimated friction coefficient will inherently depend on how the vehicle is maneuvered [13].

3.2.2 Tire Properties

When the wheel spins, the tire is deformed and recovered in repeated cycles due to the loaded weight. Hence, the friction force between the tire and the road surface has a hysteresis characteristic that is dependent on tire rubber stiffness [13]. The tire stores the deformation energy within the rubber and releases the energy in the form of a pushing force and heat resulting in a net friction force that contributes to the reduction in forward motion. Additionally, friction occurs due to adhesion which is the result of the small-scale bindings of the rubber molecules with the road surface. These types of molecular bonds are called Van der Waals bonds and are weak attractive forces lasting for a short amount of time. Thus, the total friction force F can be defined as the sum of both the hysteresis and adhesion friction forces F_H and F_A respectively as follows [13]:

$$F = F_H + F_A \tag{3.2}$$

Both the hysteresis and adhesion force components depend on the properties of the tire itself but also on the characteristics of the road surface that are described in the next section. The hysteresis effect mostly occurs on the macro-level asperities and is the dominant component on wet and rough-textured road surfaces. Conversely, adhesion occurs on micro-level asperities and is dominant on dry and smooth-textured

road surfaces [13].

3.2.3 Road Characteristics

The properties of the road surface have a direct effect on the estimated tire-road friction coefficient. The type of road surface affects the maximal friction achievable between the road and the tire. In [14], the authors studied how different road conditions affect the friction coefficient. On rough road, the friction coefficient was measured to be in the range of 0.8-0.9, whereas smooth road resulted in the range between 0.35-0.45. Finally, gravel and mud surfaces resulted in friction coefficients between 0.1 - 0.3. Hence, the type of road surface influences the friction coefficient regardless of other parameters.

3.2.4 Environmental Factors

Both temperature and precipitation have an indirect effect on the friction between tire and road surface. Water on the road itself will reduce the friction between tire and road resulting in reduced traction and control over the car. In [15], it was shown that even a 0.05 mm layer of water on the road surface can result in up to 30% less tire-road friction when compared to dry conditions. More severe cases when thicker layers of water build up on the road surface result in an increased risk of hydroplaning where the friction coefficient can drop close to zero [13].

Snow and ice are the most hazardous environmental factor when it comes to driving. In the experiments of [14], the friction coefficient of ice surfaces was below 0.1. In cases when ice or snow blocks the direct contact between tire and road surface, any sudden changes like breaking or changing direction will cause the wheels to lock and slide [13].

3.3 Machine Learning

Machine learning (ML) is a sub-field in artificial intelligence (AI) where algorithms are used to find patterns in empirical data based on how humans learn [16]. The concept of learning in ML is accomplished by gradually improving the accuracy of predictions in an iterative process commonly referred to as training. The end product of a machine learning pipeline is a trained model capable of making predictions based on new data that the model has never been exposed to before [16]. ML has proven to perform well in a variety of tasks such as classification-, regression-, clustering-, and association tasks within many different fields including self-driving cars, banking and financial services, and healthcare. In recent decades, there have been tremendous advances in the field of artificial neural networks (ANNs), especially deep neural networks (DNNs) that have proven to outperform previously state-of-the-art methods in many fields [17]. Furthermore, depending on both the available data and the task at hand, machine learning can be divided into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [18]. In this study, unsupervised- and semi-supervised learning methods were used and are therefore more thoroughly described next.

3.3.1 Unsupervised Learning

Unsupervised learning is a machine learning technique in which algorithms are trained to find patterns in unlabeled data [18]. In contrast to supervised learning where each data point is tagged with a target variable, unsupervised methods solely rely on capturing patterns in the underlying structure of the data [18]. Labeling a large amount of data can be both time-consuming and expensive but is often necessary to train models capable of making accurate predictions that reflect real-world situations. Additionally, evaluating the results of models trained in an unsupervised fashion can be challenging since there is no ground truth to compare with, especially in classification and regression tasks [19]. However, unsupervised learning is still used in a variety of tasks including data visualization, association, anomaly detection, dimensionality reduction, and data generation. Data clustering methods such as k-means, DBSCAN, and hierarchical clustering are common unsupervised learning techniques in which data points are clustered into groups based on some predefined similarity measure such as Euclidean distance, Haversine distance or cosine similarity [20]. For anomaly detection tasks, ANNs are commonly trained to find patterns in "normal data" such that anomalous data can be detected and removed [21]. An autoencoder is a type of generative ANN that was used in this study for anomaly detection and will therefore be described later in this chapter.

3.3.2 Semi-supervised Learning

Semi-supervised learning methods use a combination of both labeled and unlabeled data, where the former commonly is smaller in size [18]. Examples of semi-supervised learning techniques are active learning, self-training, and co-training. Each of these techniques involves the model to either label or pseudo-label unlabeled samples

from a larger unlabeled data set. Hence, a smaller amount of labeled data is needed which saves both time and resources. However, unlike in unsupervised learning, these models can still be trained in a supervised learning fashion using labels as ground truths to compute the loss.

Active Learning
Active learning (AL) is a semi-supervised machine learning approach used to create labeled data sets efficiently when annotating data is either difficult or expensive. There are two methods for implementing AL: pool-based and stream-based. In stream-based AL, a model is trained on an initial small labeled data set, and new samples from the unlabeled data set are then passed successively to the model that predicts a label. For certain predictions, the samples are added to the labeled data set. However, the more uncertain samples are instead labeled by the oracle (human labeler).

Pool-based active learning involves additional steps including sampling data points and using a query to select new samples as shown in Figure 3.2. The core concept in pool-based AL is that a smaller labeled data set is used to train a model to sample data from the larger unlabeled data set using a query function. The new data samples are labeled using either the model or human annotators. Finally, the labeled samples are added to the labeled data set. The model is then trained using the extended data set and the cycle is repeated until a data set of sufficient size has been achieved [22].

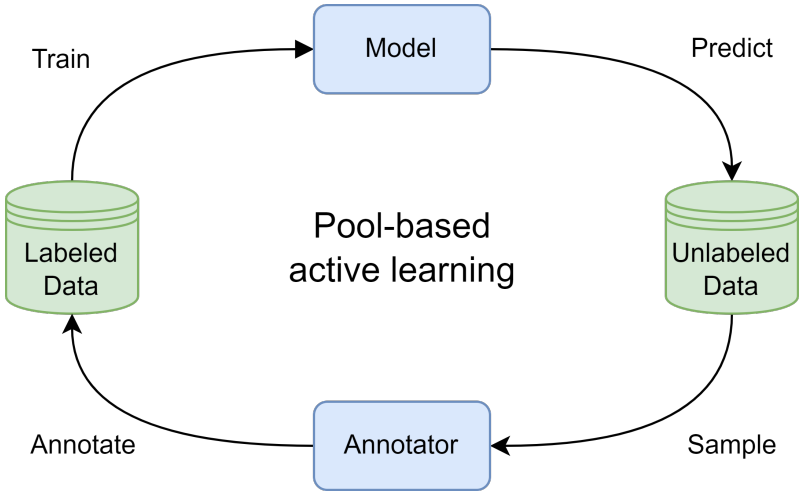

Figure 3.2: Pool-based active learning.

There are multiple methods for querying new samples in active learning including query by committee, uncertainty-based sampling, and expected model change/error reduction. The committee sampling methods score the samples using multiple models trained on the labeled data. There are several uncertainty-based methods, but the main goal is to efficiently select samples that the model struggles to classify accurately and instead manually label them. Some examples are margin-based sampling, entropy sampling, and least confidence sampling [23]. Finally, expected model change and expected error reduction are both similar and computationally heavy

methods. The change or error reduction is calculated when adding a new sample, leading to many additional computations [24].

3.3.3 Metrics

Metrics are used to measure the performance of machine learning models. For classification tasks, some of the most widely used metrics are accuracy, F1 score, precision, and recall. Accuracy can be used to assess the overall performance of the model. For binary classification tasks, accuracy can be defined as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

where: TP = True positives
TN = True negatives
FP = False positives
FN = False negatives

However, accuracy might be misleading if the data set is imbalanced i.e. when there are more instances of some class. Additionally, accuracy is not a good measure of the model's performance if there are different costs for false positives and false negatives. Therefore, other metrics such as F1-score are usually more useful than accuracy. F1-score is computed based on the precision and recall of a model. Precision measures the proportion of positive predictions that were correct and is computed as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.4}$$

In contrast, recall measures the completeness of positive predictions and is computed as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.5}$$

Finally, with precision and recall, the F1-Score is computed as follows:

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.6}$$

3.3.4 Hyperparameter Tuning

Hyperparameter tuning is essential when optimizing the performance of a machine learning algorithm. A hyperparameter is a parameter that directly affects the learning process including batch size, learning rate, and number of epochs. The optimal hyperparameters for a model differ depending on the data that the model is being trained on. Furthermore, the goal in hyperparameter optimization is to find a combination of hyperparameters that yields the optimal value of an objective function e.g. loss or accuracy [25].

The most common approaches for hyperparameter tuning are grid search and random search. Grid search is an exhaustive search method in which the best-performing model is assessed by training the model on each combination of provided hyperparameters. In random search, the combination of hyperparameters is randomized which in some cases can lead to more optimal hyperparameters with less number of iterations. Even though both grid- and random search often can be computed in parallel, they are time-consuming methods, especially for large numbers of hyperparameters [25].

Another approach for hyperparameter tuning is to use hyperparameter optimization frameworks such as Optuna which is designed to find the most optimal parameters automatically and efficiently. A benefit of using Optuna is that the search space is dynamically constructed using a sampling strategy. Both grid search and random search can be used as the sampling method in Optuna. Additionally, other model-based sampling methods available in Optuna are tree-structured Parzen estimator (TPE), Gaussian processes (GP), and covariance matrix adaptation evolution strategy (CMA-ES). In this thesis, the TPE sampling method was used which is based on independent sampling of previously evaluated hyperparameters and is known to generally perform well. Another benefit of using Optuna is that trials can be terminated if they do not seem promising. This means that more time can be spent on better trials during the search for the optimal hyperparameters. Finally, after the hyperparameter tuning process has finished, Optuna has the option to extract the most important hyperparameters that had the most impact on minimizing or maximizing the objective value [26].

Additionally, Optuna can handle multiple objectives during the hyperparameter tuning process. In multi-objective optimization, it is common to end up with more than one optimal combination of hyperparameters and trade-offs must therefore be made. Finally, with Optuna it is possible to simply extract the most optimal trials for multiple objectives by computing the Pareto front [27].

3.4 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a clustering algorithm used to group spatial data points by how densely packed they are [28]. Points that are closely packed together will form a cluster while other points located in low-density regions will be classified as outliers. Hence, DBSCAN can detect outliers and handle noisy data well unlike other famous clustering algorithms such as K-means. The main idea of DBSCAN is to cluster the given data based on a concept called ϵ-neighbour that is defined as follows:

Definition 1 *Let D be a data set containing points. Two points $p, q \in D$ are neighbours if their distance is smaller than the radius ϵ around p:*

$$N_\epsilon(p) = \{q \in D \mid d(p, q) < \epsilon\}$$

where d is the distance measure.

Examples of distance measures are Euclidean-, Manhattan-, and Haversine distances. The latter is defined as the angular distance between two points (longitude and latitude) on the surface of a sphere. Unlike Euclidean- and Manhattan-distance, the Haversine distance takes the curvature of the sphere into account. In DBSCAN, the radius ϵ is a value that determines the maximum distance between two points for one of the points to be considered as a neighbor to the other point. Choosing a too-small epsilon would lead to many points being classified as outliers. Conversely, a too-high epsilon value would result in merged clusters that should have been separated. Another tunable parameter in DBSCAN is a value determining the minimum number of points within the radius ϵ needed for the center point to be considered as a core point.

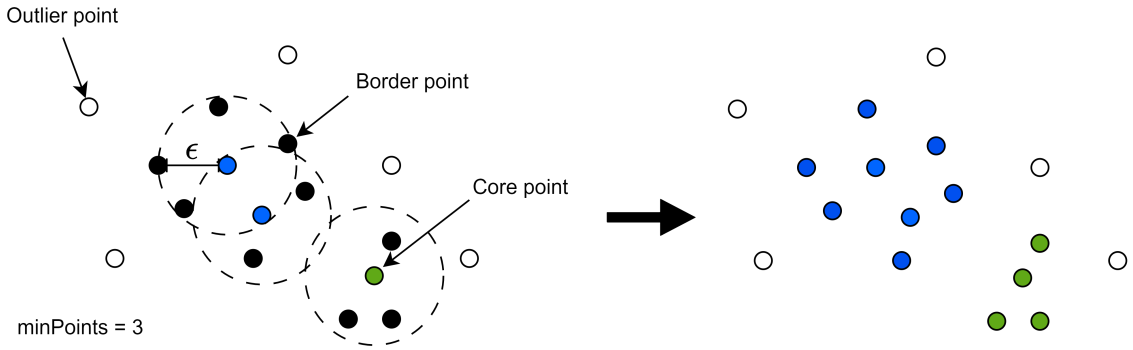

Figure 3.3: How DBSCAN works.

DBSCAN starts by defining core points as shown in Figure 3.3. Core points are used to extend the clusters further. Ultimately, when all core points have been assigned a cluster, non-core points or border points that are within the ϵ radius are added to the cluster. Border points can only be added to clusters and not be used to add new points to the clusters. Finally, the remaining points are considered to be outliers since they are not close to any core point.

3.5 Artificial Neural Network

An artificial neural network (ANN) is a machine learning algorithm inspired by the biological neural network that exists in the human brain [29]. Similar to the human brain, ANNs are built upon a set of interconnected nodes/neurons that send and receive signals in a complex system. In an ANN, these signals are real numbers that are being processed by the nodes to form an output that is computed by a non-linear function, called activation function. The simplest ANN is a single-layer perceptron consisting of multiple inputs and an output, as shown in Figure 3.4.

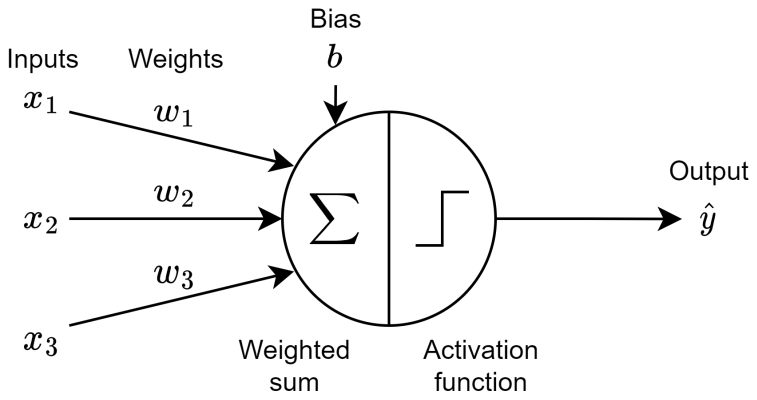

Figure 3.4: A single layer perceptron with three input variables and one output.

A perceptron with n input variables is a type of linear classifier that computes the output $\hat{y}$ in two steps. First, each input variable $x_1, x_2, \ldots, x_n$ is multiplied with its corresponding weight $w_1, w_2, \ldots, w_n$, as shown in Equation 3.7.

$$z = \sum_{i=1}^{n} x_i w_i + b \tag{3.7}$$

Additionally, a bias term b is added to this weighted sum that serves as an additional parameter that can be tuned to improve the performance of the perceptron. Second, the weighted sum is passed through an activation function f that determines if the node should be activated or not, as shown in Equation 3.8.

$$\hat{y} = f(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

Common activation functions are sigmoid, tanh, and rectified linear unit (ReLU) [30]. In this simple example, the activation function is a function that activates if the input is positive and deactivates if the input is negative. Hence, the output of this perceptron will be binary (0 or 1).

The single-layer perceptron can handle linear classification tasks but struggles when the data is not linearly separable [29]. Therefore, a multi-layer perceptron (MLP) also referred to as a feed-forward neural network (FNN) was developed with the idea

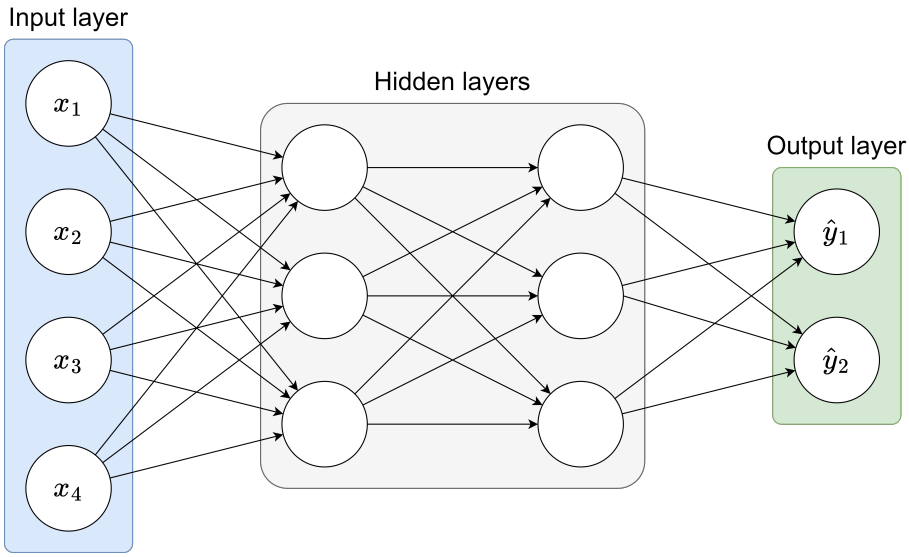

Figure 3.5: A multi-layer perceptron.

of creating a non-linear model by combining multiple single-layer perceptrons in a network. An MLP is a fully-connected neural network meaning that each neuron in one layer is connected with each neuron in neighbouring layers, as shown in Figure 3.5. A connection between two neurons enables the output signal from one neuron to be passed as input to the next neuron. The layers can be divided into an input layer, hidden layers, and an output layer. Both the input- and output layers contain only a single layer. However, between these layers are the hidden layers that can contain any number of layers and neurons. When calculating the depth of an ANN, only the layers with adjustable weights are considered, i.e. the hidden layers and output layer. Increasing the number of hidden layers increases the number of parameters and complexity of the network. Hence, training a network with many layers will be more time-consuming and the network might overfit to the training data and thus perform worse when exposed to new data. Conversely, a network containing too few hidden layers will struggle to learn the underlying representation of the data. ANNs containing many layers are referred to as deep neural networks (DNNs) and will be discussed later in this chapter. The network shown in Figure 3.5, has four neurons in the input layer for each input. The input data is then passed to the two hidden layers containing three neurons each. Finally, the signals from the last three neurons in the last hidden layer are passed to the output layer containing two neurons to form the output.

Similar to humans, ANNs learn by making initial guesses and improving these guesses when new information is given and processed. The goal of training an ANN is that it should learn how to make accurate predictions by finding the optimal weights and biases for each neuron in the network. Training an ANN is an iterative process that consists of two main concepts: forward- and backpropagation. In forward propagation, the input data is passed forward to each neuron in the network. As previously mentioned, the outputs of each neuron in the network are computed using an activation function that suppresses the input values to a bounded range

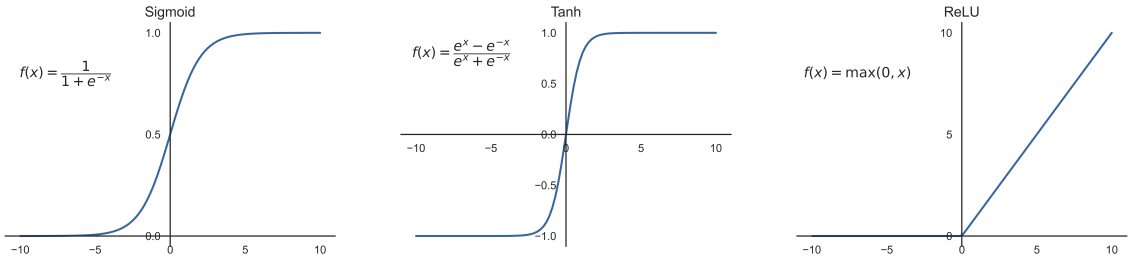

Figure 3.6: Three common activation functions.

such that the problem of exploding gradient is prevented. The most commonly used activation functions are sigmoid, tanh, and ReLU as shown in Figure 3.6.

The sigmoid activation function is a non-linear function that transforms the input to a value between 0 and 1, and is thus suitable for producing probabilities [30]. Since many loss functions expect the input values to be in the range of 0 to 1, the sigmoid activation function is commonly used for neurons in the output layer. The tanh function is another non-linear function that pushes the input values to the range -1 to 1. Both the sigmoid and tanh activation functions are differentiable everywhere meaning that both activation functions can be used for gradient-based backpropagation. However, both functions can cause the vanishing gradient problem [30]. This means that the weights and biases are updated with small changes resulting in slow learning. The ReLU activation function is commonly used in the hidden layers instead of sigmoid and tanh to fix this problem. Unlike sigmoid and tanh, ReLU is not differentiable at zero. This means that a sub-derivate is needed in the backpropagation when using ReLU as an activation function. Additionally, ReLU introduces sparsity in the hidden layers which have been proven to be beneficial in ANNs and are achieved when some values are negative and converted to zero by the ReLU function. However, ReLU suffers from the dying ReLU problem that occurs when too many activations get below zero resulting in the deactivation of neurons and thus a reduction in learning [31].

The final output from the neurons in the output layer is a predicted vector $\hat{y}$ that is compared with the target vector y by using a loss function. The loss is a measure of how far the predicted value is from the target and the choice of loss function depends on the type of task the network tries to solve e.g. classification or regression. For binary classification tasks, the binary cross entropy (BCE) is commonly used and is computed as follows [32]:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)) \tag{3.9}$$

The BCE loss function expects the predicted values $\hat{y}_i$ and target values y_i to be interpreted as probabilities ranging from 0 to 1. Additionally, BCE loss is differentiable at all points making it suitable for gradient computations. Another commonly used loss function for binary classification tasks is the hinge loss defined as follows [33]:

$$L_{HINGE} = \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - (\hat{y}_i \cdot y_i)) \tag{3.10}$$

Similarly to BCE, the hinge loss function also expects the predicted values to be between 0 and 1. However, the hinge loss is not differentiable at one meaning that sub-derivatives are needed. For regression tasks, mean absolute error (MAE) also known as L1 loss is a commonly used loss function and is calculated by taking the absolute value of the difference between the predicted- and target values as follows [34]:

$$L_{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.11}$$

Another loss function for regression tasks that is similar to MAE is the mean squared error (MSE) that is computed by taking the square of the difference between the predicted- and target values [35].

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.12}$$

Unlike MAE, MSE penalizes large errors since the difference is squared. If many predicted values result in low losses, then MSE is preferable. However, if the model makes a few poor predictions, the MSE loss will magnify the error due to the squaring of the difference. This will result in an overall high error using MSE even though the majority of predictions were accurate. In that case, MAE might be the better alternative.

The goal of training an ANN is to reduce the loss by updating the weights and biases of the network during backpropagation. Finding the smallest loss is an optimization problem in multi-dimensional space tackled by the use of an optimizer. Most of the optimizers are based on gradient descent which is an algorithm that converges to a local minimum of the loss function. Some of the most well-known optimizers are stochastic gradient descent (SGD), root mean squared propagation (RMSprop), and adaptive moment estimation (Adam) [36]. During backpropagation, the weights w_i and biases b of the network are updated using gradient descent as follows [37]:

$$\begin{aligned} w_i &= w_i - \eta\frac{\partial L}{\partial w_i} \\ b &= b - \eta\frac{\partial L}{\partial b} \end{aligned} \tag{3.13}$$

The learning rate η is a tunable hyperparameter that determines the size of each step taken towards a local minimum of the loss function in gradient descent [37]. A too high learning rate might result in overshooting the local minimum. Conversely, a too low learning rate can result in slow convergence or possibly being stuck at an undesirable local minimum. Commonly, the learning rate is set to a value between 1×10^{-3} and 1×10^{-5} but needs to be tuned for optimal performance.

3.5.1 Recurrent Neural Network

A recurrent neural network (RNN) is a type of artificial neural network (ANN) that handles sequential data as input [38]. Similar to vanilla ANNs, RNNs consist of an input layer, hidden layers, and an output layer. However, traditional ANNs do not take the order of the input data into account and can therefore not find temporal or structural patterns in sequences. Therefore, RNNs use an additional memory mechanism in the form of a feedback loop that allows the model to reuse past information in the computations of the next output [38] as shown in Figure 3.7. RNNs can be used for different types of sequence modeling tasks including sentiment classification, text generation, image captioning, translation, or forecasting. For classification tasks, the output of the RNN will not be a sequence but rather a vector of probabilities. For sequence generation tasks, commonly referred to as sequence-to-sequence or seq-to-seq, the relationship between the input and output is many-to-many since both the input and output are a sequence [39].

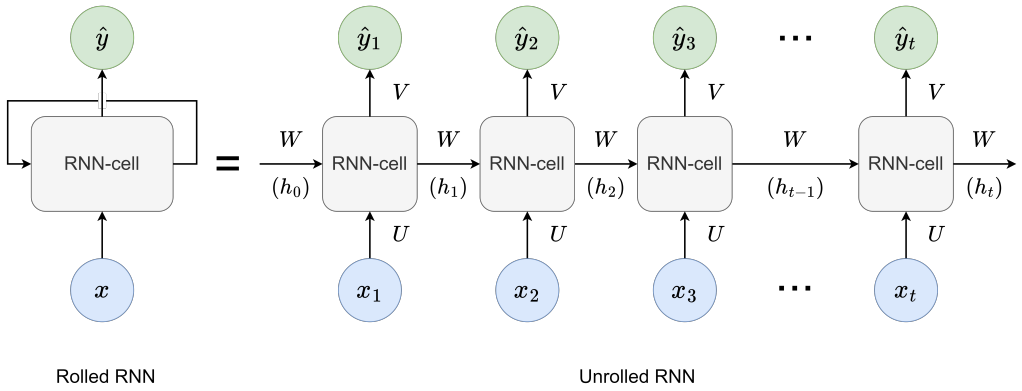

Figure 3.7: The unrolling of a recurrent neural network.

The RNN is unrolled across time into a full network for each element in the sequence. An RNN cell is similar to a neuron in a feed-forward neural network in that it computes an output vector by the use of an activation function on the weighted sum of the inputs. However, in addition to the output vector, RNN cells produce a hidden state h_t that is passed to the next RNN cell in the network such that the output is influenced by both current and previous values. The weights and biases are shared across time meaning that the number of parameters does not change with increasing number of unrolls of the network. During forward propagation of the network, the hidden state h_t and output y_t is updated for each time step t as follows [40]:

$$\begin{aligned} z_t &= Wh_{t-1} + Ux_t + b \\ h_t &= \tanh(z_t) \\ o_t &= Vh_t + c \\ \hat{y}_t &= \text{softmax}(o_t) \end{aligned} \tag{3.14}$$

where: U = input-to-hidden weight matrix
V = hidden-to-output weight matrix
W = hidden-to-hidden weight matrix
b, c = bias vectors

During the backpropagation of an RNN, the gradients flow backward for each time step and across all time steps starting from time t. This process is referred to as backpropagation through time (BPTT) [38]. The gradients are computed with respect to each weight matrices U,V,W, and bias vectors b and c. Each parameter is then updated similarly to a regular ANN. Due to the unrolling of the RNN, the network usually becomes deep, resulting in many repeated gradient computations. This can in turn result in vanishing or exploding gradients, i.e. gradients becoming too small or too large respectively [38]. Hence, the network will not be able to update the weights and biases properly and the learning will be reduced or stopped entirely. In other words, vanilla RNNs suffer from short-term memory and struggle to carry information from earlier time steps, especially for long sequences. A solution to exploding gradients is gradient clipping where large gradients are clipped back to a feasible scale. Vanishing gradients is a more common problem in RNNs and can somewhat be mitigated by choosing a suitable activation function, initializing the weights, or changing the network architecture [38].

3.5.2 Long Short-Term Memory

As previously described, vanilla RNNs are commonly facing the vanishing- and exploding gradient problem that inherently prohibits the network from learning. To counteract this issue, long short-term memory (LSTM) cells were introduced with the idea of including both long- and short-term memory when computing the outputs of each cell in the RNN [41]. The long-term memory of an LSTM cell is represented by the cell state c that flows through the top of the LSTM cell in Figure 3.8. In contrast, the short-term memory is reflected in the hidden state h. The main idea of LSTMs is that the gradients for the cell states neither vanish nor explode since they are not directly computed w.r.t the weights. LSTMs are more complex compared to regular RNNs but are better at learning the inter-dependencies in sequential data [41] through the use of gates.

The first gate of an LSTM cell is the forget gate where the input vector x_t at the current time step t and the previous hidden state h_{t-1} is passed through a sigmoid activation function that outputs a value ranging from 0 to 1 (probability) as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.15}$$

The first hidden state is usually initialized by setting each value to zero. The output from the sigmoid activation function is then multiplied with the previous cell state c_{t-1}. Hence, the output of the sigmoid activation function determines the amount of long-term memory that should be forgotten [41].

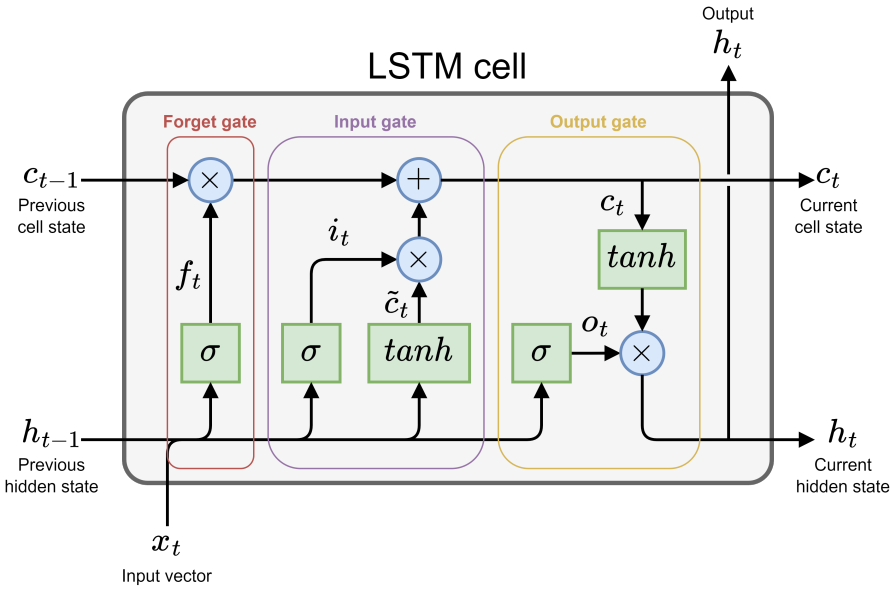

Figure 3.8: Long short-term memory (LSTM) cell.

Next, in the input gate, the previous hidden state and current input are passed to both a sigmoid- and a hyperbolic tangent activation function as follows:

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \tilde{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \end{aligned} \tag{3.16}$$

The latter is used to compute potential long-term memory while the former is used to decide the percentage of long-term memory to be added to the cell state. The current cell state c_t will be updated based on the outputs from both the forget gate and input gate as follows [42]:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.17}$$

The final gate in an LSTM cell is the output gate where the values of the current cell state c_t are passed to a hyperbolic tangent function to compute potential short-term memory. Similar to the other gates, the sigmoid activation function is used on the previous hidden state and the current input to compute a probability that determines the amount of short-term memory to be passed to the next LSTM cell as follows [42].

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{3.18}$$

3.5.3 Gated Recurrent Unit

A gated recurrent unit (GRU) is a type of RNN that is similar to LSTM in that gates are used to update the hidden states of the network for each time step. GRUs are

less complex compared to LSTMs which makes them less computationally expensive while still maintaining a high performance comparable to the performance of LSTMs. Unlike LSTMs, GRUs consists of only two gates and do not use the internal cell state, i.e. the long-term memory state. Hence, in theory, LSTM networks should outperform GRUs in cases when the input sequences are long [43].

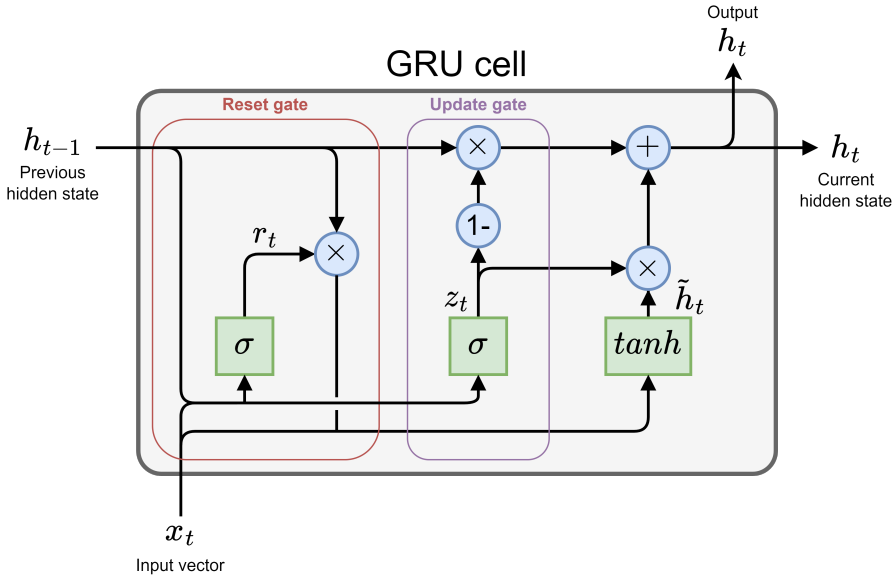

Figure 3.9: Gated recurrent unit (GRU) cell.

A GRU cell consists of a reset- and update gate as shown in Figure 3.9. The input vector x_t and previous hidden state h_{t-1} are first passed to the sigmoid function in the reset gate as follows [43]:

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{3.19}$$

The sigmoid function outputs a value between 0 and 1. This value is used later in the update of the current hidden state h_t and determines how much of the previous information should be forgotten. Next, the input vector x_t and previous hidden state h_{t-1} are passed to the sigmoid function in the update gate where the output is computed as follows [43]:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{3.20}$$

This value is used to update the hidden state. The candidate hidden state $\tilde{h}_t$ is computed using a hyperbolic tangent function. The output value from the sigmoid function in the reset gate r_t is multiplied with the previous hidden state h_{t-1}. This product is used in the weight matrix multiplication. Additionally, the input vector x_t is also multiplied with the weight matrix. The bias term is added and the entire matrix is passed to the hyperbolic tangent function that outputs the candidate hidden state $\tilde{h}_t$ as follows [43]:

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) \tag{3.21}$$

Furthermore, to compute the current hidden state h_t, the candidate hidden state is multiplied with the output from the sigmoid function in the update gate z_t. The opposite or complement probability of z_t is computed and multiplied with the previous hidden state h_{t-1}. The sum of these element-wise products forms the current hidden state as follows [43]:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.22}$$

3.5.4 Autoencoder

An autoencoder (AE) is an unsupervised learning technique based on an artificial neural network that is trained to reconstruct high-dimensional input data. Autoencoders can be used in a variety of applications including feature extraction, dimensionality reduction, image generation, and anomaly detection. Autoencoders for anomaly detection is a special use case where the reconstruction error or loss is used to classify and filter out anomalies [3]. Some examples of applications where autoencoders are used for anomaly detection are [44], [7], [6].

The structure of an autoencoder consists of three main parts: encoder, bottleneck, and decoder, as shown in Figure 3.10. The encoder can be viewed as a function $g(x)$ that can transform the input data x into a latent vector representation z. The encoded vector z is located in the bottleneck of the network and is a compressed representation of the input data. This vector is then up-scaled to the same dimension as the input data in the decoder part $f(z)$ of the network. Ideally, after training the autoencoder, the input x should be identical to the reconstructed output $\hat{x}$ [45].

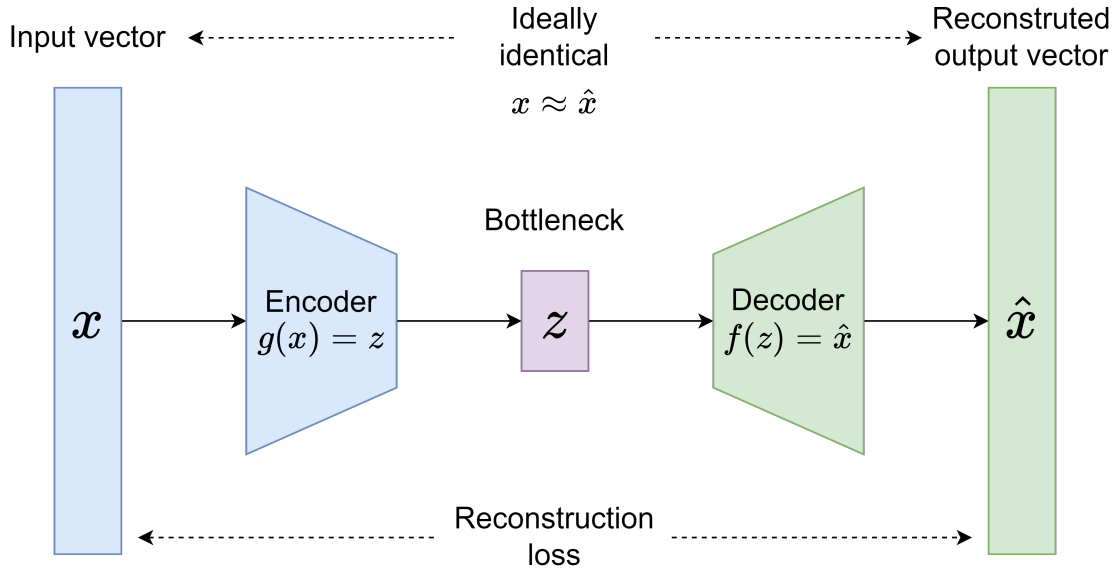

Figure 3.10: The structure of an autoencoder.

As previously mentioned, autoencoders can be used for anomaly detection tasks. The main idea is to train the autoencoder on normal data that contains a majority of normal data points. Ideally, this data set should not include any anomalies. However, this can be challenging to achieve, especially when the causes of the anomalies are unknown. Furthermore, the autoencoder will encode the normal data into a latent vector representation that is decoded back to its original dimension. The autoencoder is trained to minimize the reconstruction loss such that the reconstructed output will be similar to the input after training. Hence, smaller reconstruction losses will in general correspond to reconstructed outputs that are more similar to the input. After training, the autoencoder should have learned the latent distribution of the input data such that when anomalous data is inputted, the reconstruction loss will be higher compared to when normal data is inputted. A threshold on the reconstruction loss can therefore be used to classify if the inputted data is normal or anomalous.

Autoencoders for anomaly detection tasks can be summarised in the following four steps:

1. Train the autoencoder on normal data.

2. Feed normal data as input to the trained autoencoder and compute the distribution of the reconstruction loss.

3. Select a threshold on the reconstruction loss e.g. such that the majority of normal data is being classified as normal.

4. Test and evaluate the trained autoencoder with the given threshold by observing the number of anomalies being detected.

One of the most important hyperparameters for autoencoders is the embedding size that determines the size of the compressed output vector of the encoder. The embedding size can have a large impact on the model's ability to distinguish false- from true positive sequences. A too large size of the embedding will enforce the autoencoder to learn the identity function such that the model will generate low losses for any input and thus make the model unreliable in detecting anomalies [46]. Conversely, a too small embedding size will make it too difficult for the model to reconstruct the data since too much information is lost.

Traditionally in machine learning, the objective of hyperparameter tuning is to minimize the validation loss or maximize the accuracy. However, finding the most optimal hyperparameters for autoencoders in the setting of anomaly detection is a non-trivial task because the loss is not a rigorous measure of the performance on its own. For example, increasing the embedding size will result in the input data not being compressed much and the decoder will easily decode the latent vector representation and reconstruct the input data such that the reconstruction loss will be small for all inputs. However, the autoencoder will struggle to find and learn patterns specific to the data it is being trained on and thus will not perform well in detecting anomalies [46].

3.6 Spatial Hashing

Spatial hashing is a method that is used to reduce the number of comparisons needed to find neighbouring points [47]. Assume the goal is to map each point p_i from the data set $D_1 = \{p_1, p_2, \ldots, p_n\}$ with each point q_j from the data set $D_2 = \{q_1, q_2, \ldots, q_m\}$ based on their spatial relationship. A naive solution would be to compare the locations of every point with each other. This results in a time complexity of $O(nm)$, where n and m are the numbers of points in D_1 and D_2 respectively. Spatial hashing is a significantly more efficient solution, especially with an increasing number of points in the data sets. The main idea with spatial hashing is to reduce the number of comparisons needed to find neighbouring points in two steps. First, each point $p_i \in D_1$ is assigned to cells in a grid using a hash function as shown in Figure 3.11. The hash function can be used to e.g. map latitudinal- and longitudinal coordinates to cell indices in 2D or 3D space [47].

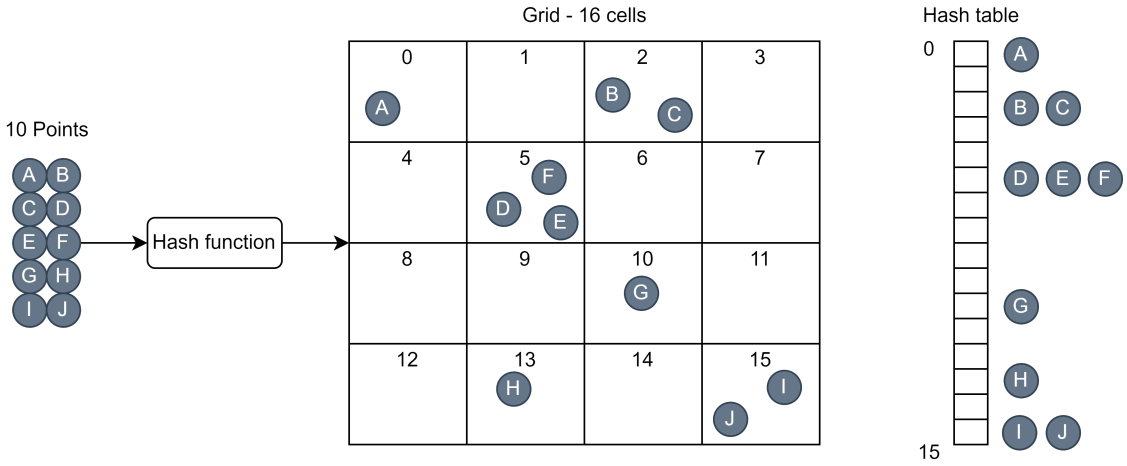

Figure 3.11: Spatial hashing.

Furthermore, to find neighbouring points of a point $q_i \in D_2$, the point is hashed to a cell index using the same hash function. The distance is computed between q_i and each point from D_1 that was assigned to adjacent cells, including the cell to which q_i was assigned. Hence, instead of computing the distance between every point as in the naive solution, the point q_i is only compared with points in neighbouring cells. The cell size influences the number of points that are mapped to the same cell and is a parameter that needs to be tuned for optimal performance. A too large cell size would result in too many points being hashed into the same cell and thus increase the number of comparisons needed. Conversely, a too small value would lead to points spanning too many cells which can result in points not being compared as they could get grouped into grids further apart [47].

The time complexity of spatial hashing depends on how the points are disturbed in the cells. In cases when all points are hashed to unique cells, the complexity is $O(1)$. The worst case is when all points are assigned the same cell which results in the time complexity of $O(nm)$, which is the same as for the naive approach [47].

4

Data Description and Analysis

This chapter provides a description and analysis of the available data for this project. Section 4.1 presents the available data in this study which originated from three sources: Volvo Cars Corporation (VCC), Trafikverket, and SMHI. This is followed by an analysis of the data in Section 4.2. Since the data was unlabeled, the analysis was essential to find potential root causes of false positive alerts and to justify choices made in the methodology.

4.1 Available Data

The available data used in this study is divided into the following four sets: vehicle data, slippery road alert (SRA) data, weather data, and road data. The vehicle- and SRA data originated from Volvo Cars Corporation (VCC) while the weather- and road data were requested and gathered from Swedish Transport Administration (Trafikverket). Additionally, the weather data used in the analysis was gathered from Swedish Meteorological and Hydrological Institute (SMHI).

4.1.1 Vehicle Data

The vehicle data was gathered from the cloud for 10 weeks (72 days) between the 12th of January and to 24th of March 2023 during winter and early spring in Gothenburg, Sweden. Ultimately, after the data gathering process, the vehicle data contained approximately 115 million data points. Due to GDPR, the vehicle data was anonymized such that unique vehicles could not be identified.

The most important features of the vehicle data that were used in this thesis are presented in Table 4.1. This data contained bursts of data points sent from Volvo vehicles across Gothenburg and was transformed into sequences in the preprocessing step that is described in Chapter 5. The sequential vehicle data was then used to train and evaluate the machine learning models.

Table 4.1: Features of the vehicle data.

	Feature	Abbreviation	Data type
1	Friction coefficient	FC	float
2	Friction quality	FQ	int
3	Creation time	CT	timestamp
4	Coordinate	COORD	geolocation
5	Temperature	T_v	float
6	Wiper speed	WS	int
7	Vehicle identification number	VIN	string
8	Software versions	SV	string
9	Road max speed	RMS	int

1. Friction coefficient
The estimated friction coefficient was the main feature of interest in this thesis and is a value ranging from 0 to 1, where 0 represents no friction and 1 represents maximum friction. Friction coefficients below or equal to 0.35 are defined as low friction while friction coefficients above 0.35 are defined as high friction. This estimated value is based on the longitudinal brush model, lateral tire-force model, and limit handling measurement based on the ABS and traction control system.

2. Friction quality
The friction quality is an integer number ranging from 4 to 7 indicating the quality of the estimated friction coefficient. The calculations of this value are based on the variance of slip and a multiple of in-signals to the vehicle such as the lateral- and longitudinal forces on the wheels.

3. Creation time
The creation time represents the time in which the friction data was created on the database and is expressed by the date and time ranging from hours to milliseconds.

4. Coordinate
The coordinates in the friction data correspond to the longitude- and latitude coordinates of each data point.

5. Temperature
The temperature measurement originates from the vehicle temperature sensor and is represented in Celsius degrees.

6. Wiper speed
The wiper speed is an integer value ranging from 0 to 6 indicating the speed of the car wiper. This value correlates with the precipitation. However, since the wiper speed can be set manually, it is not an accurate nor reliable indicator of precipitation.

7. Vehicle identification number
The vehicle identification number (VIN) is a unique identification number for each vehicle and was only used for creating the sequential data. This number was removed and was not present in the data sets due to GDPR.

8. Software versions
Software versions represent multiple versions of the software in the electronic control unit (ECU).

9. Road max speed
The maximum legal speed for the road ranges from 5 km/h to 120 km/h.

Figure 4.1 shows a sample of data points from the vehicle data plotted on a map. Each data point corresponds to a row in the vehicle data with each of the previously mentioned features as columns. As can be seen in the upper right corner, vehicles in motion send bursts of data points to the cloud resulting in the formation of sequences with both temporal and spatial properties. This was an important discovery that was exploited in the creation of the data sets for the machine learning models.

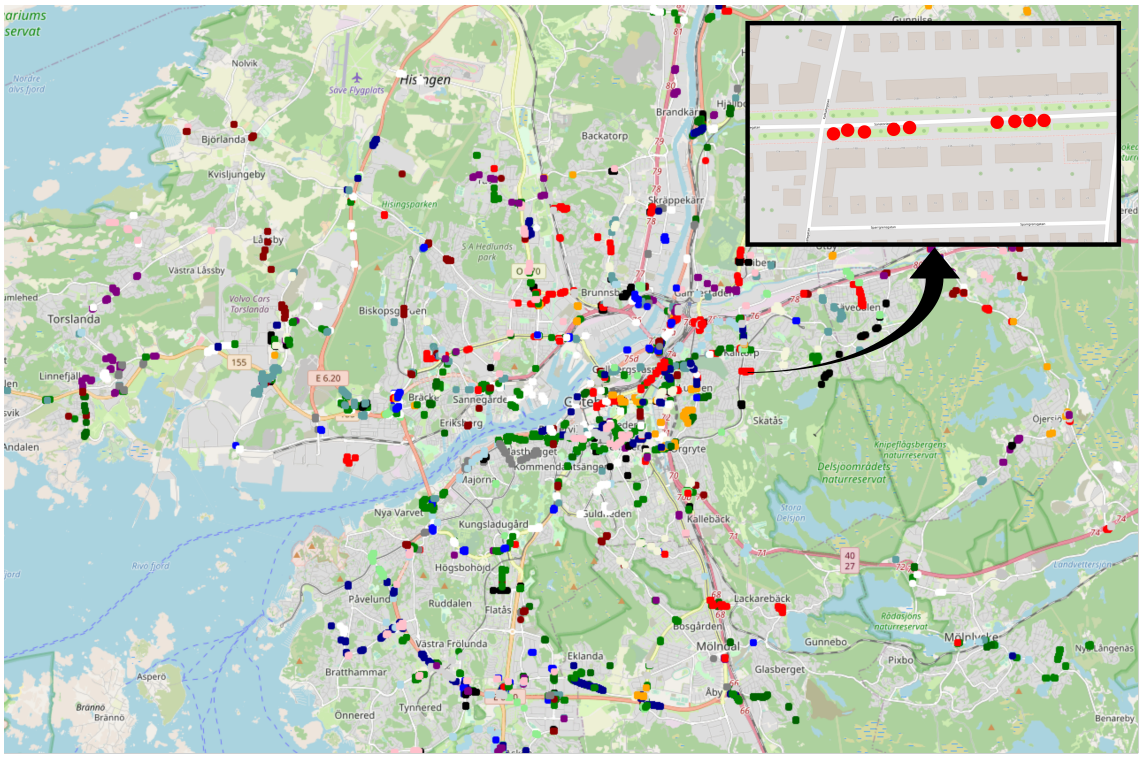

Figure 4.1: Sample of data points from the vehicle data plotted on a map.

4.1.2 Slippery Road Alert Data

The slippery road alert (SRA) data consisted of both location and time of specific SRAs generated in Gothenburg, Sweden within a year from 25th of March 2022 to 24th of March 2023. This data was only used in the analysis to find root causes of when potential false positive SRAs can occur. Furthermore, the SRA data was previously aggregated meaning that unique vehicles could not be identified in this data.

The features of the SRA data are shown in Table 4.2. The start- and end times are expressed by the date and time ranging from hours to milliseconds. The locations of the SRAs are indicated by the longitudinal- and latitudinal coordinates.

Table 4.2: Features of the slippery road alert data.

	Feature	**Data type**
1	Start time	timestamp
2	End time	timestamp
3	Location	coordinate

4.1.3 Weather Data

The weather data used to create the data sets was requested from Trafikverket and consisted of both temperature and precipitation measurements from 11 weather stations located near roads scattered across Gothenburg, Sweden. The features of this weather data are shown in Table 4.3. The station id was an integer value corresponding to the 11 weather stations. The road temperature was measured in Celsius degrees and the road precipitation level was measured in millimeters. The time interval of this data was 30 minutes. This data was used to filter the vehicle data to form the data sets used for training and testing the machine learning models. Additionally, this data was also used to train some of the models using the active learning approach.

Table 4.3: Features of the weather data from Trafikverket.

	Feature	**Abbreviation**	**Data type**
1	Station id	SI	int
2	Date and time	DT	timestamp
3	Road temperature	T_r	float
4	Road precipitation	P_r	float

The weather data used in the data analysis originated from SMHI. This data was used since the weather data from Trafikverket was not available at the time when the analysis was conducted. Additionally, the weather data from SMHI was daily based and computed using more sophisticated algorithms than just computing the mean for each day, which would be the alternative if the weather data from Trafikverket were to be used instead. The temperature was measured in Celsius degrees while the

precipitation levels were measured in millimeters. The data from SMHI originated from a weather station in Gothenburg and was gathered via SMHI's publicly open website. The features of the weather data from SMHI are presented in Table 4.4.

Table 4.4: Features of the weather data from SMHI.

	Feature	Data type
1	Date and time	timestamp
2	Daily air temperature	float
3	Daily precipitation	float

4.1.4 Road data

The road data was gathered from Trafikverket's site Lastkajen [48] within the area of Gothenburg. Lastkajen is a web-based application open to the public where one can order and download Sweden's road and railway data. For this thesis, the most valuable road data was speed bump locations, and road types as shown in Table 4.5. The speed bump locations consisted of single longitudinal- and latitudinal coordinates for each location of speed bumps whereas the road types were represented by polygons of multiple longitudinal- and latitudinal coordinates.

Table 4.5: Features of the road data.

	Feature	Data type
1	Speed bump location	coordinate
2	Asphalt road	coordinates
2	Gravel road	coordinates

4.2 Data Analysis

An initial data analysis of the unprocessed data from VCC and Trafikverket was conducted for three main reasons. First, since the available data was unlabeled, this initial analysis was essential to get an intuition of in which circumstances true- and false-positive warnings might occur. A hypothesis was that most of the generated alerts during a year are true positives and occur when the probability of ice formation on the road surfaces is high, i.e. when the precipitation levels are high and the temperature is below 0 $°C$. Conversely, some false positive alerts might occur when driving on speed bumps or when making a sharp turn for example. Second, the insights from this analysis were used to motivate the choices made in this project including how the raw data was processed and how the models were constructed. Third, this initial analysis might reveal correlations between features and limitations of the data that need to be taken into consideration in the creation of the final data sets used to train and test the machine learning models. Ultimately, the goal of this initial data analysis was to answer the following questions:

- Are there seasonal patterns in the number of generated slippery road alerts?
- What are the root causes of true- and false-positive slippery road alerts?
- In which weather conditions are low- and high values of the estimated friction coefficient sent to the cloud?

The majority (56%) of the alerts were generated during spring (March to May), as shown in Figure 4.2. This was somewhat unexpected since winter is commonly more associated with slippery roads. However, further investigation into the data reveals that the maximum number of alerts occurred during a week in spring when it snowed heavily and the temperature was below 0 $°C$. Furthermore, the number of alerts during winter was 34% while 7% occurred during autumn and only 3% occurred during summer.

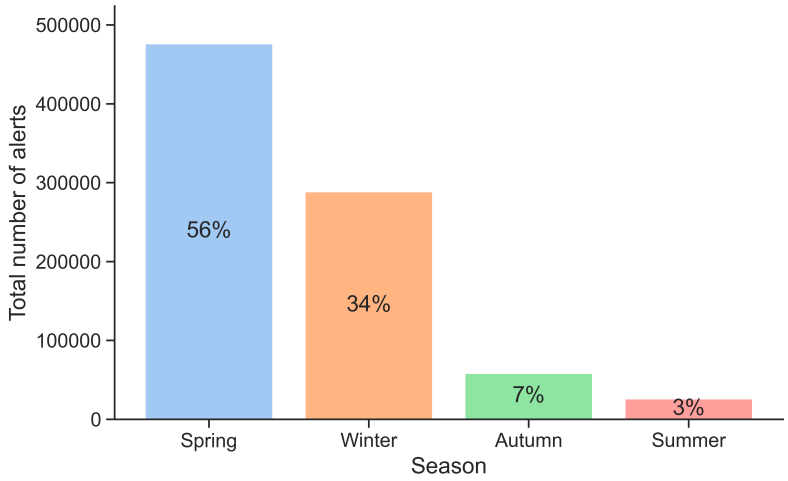

Figure 4.2: Number of generated alerts for each season within a year.

Most of the generated alerts during spring and winter are believed to be true pos-

itives caused by ice formation on the road surfaces. Hence, finding false positive alerts during these seasons can be challenging. On the contrary, most of the alerts generated during autumn and summer are probably false positives. However, both true- and false positives can occur in any season. For example, some true positives during summer and autumn could be caused by hydroplaning. Conversely, during spring and winter, hard braking or driving on a speed bump can potentially cause false positive alerts.

The vehicle data used to create the data sets for the machine learning models were gathered during late winter and early spring. Hence, the number of generated alerts from the SRA data during this period is of high interest. Figure 4.3 shows the number of generated alerts (orange) against the air temperature (grey) and precipitation levels (blue) for each day from 2023-01-12 (winter) to 2023-03-24 (early spring).

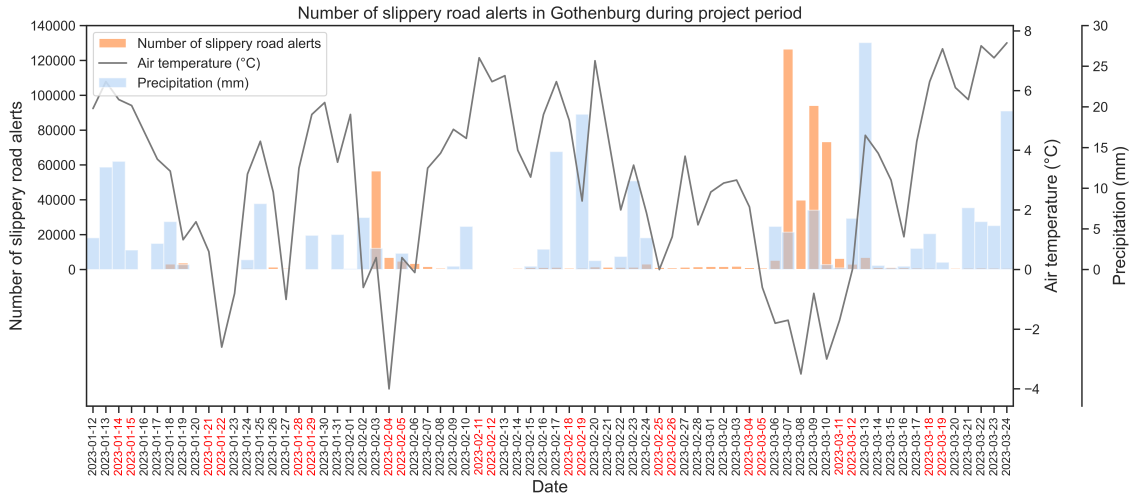

Figure 4.3: Number of generated slippery road alerts in Gothenburg during the project (late winter and early spring).

As expected, when the daily air temperature dropped below 0 °C and the precipitation levels were above 0 millimeters, then the number of alerts increased. Note that the daily air temperature dropped below 0 °C four times within this period. The first two drops occurred on the 22nd of January and 27th of January. Even though the daily air temperature was below 0 °C, none or few alerts were generated. The reason for this is probably because of no precipitation. The next drop in air temperature below 0°C occurred on the 4th of February when the precipitation levels were above 0 millimeters. As expected, this increased the number of generated alerts. Similarly, the number of generated alerts increased in the final drop in air temperature which occurred around the 8th of March. The precipitation levels were above 0 millimeters during this period as well. The maximum number of alerts (approximately 130 000) occurred on the 7th of March. In general, when the daily air temperature was above 0 °C, the number of generated alerts was few. Some of these alerts might be false positives. An increase in precipitation levels during those days did not seem to drastically increase the number of generated alerts.

The number of daily generated alerts during summer 2022 in Gothenburg is shown

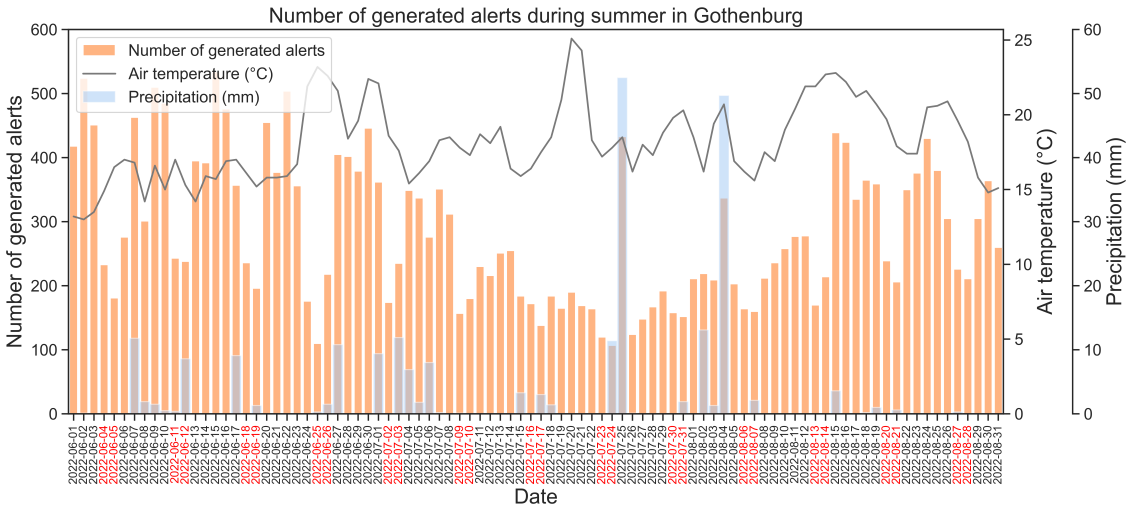

Figure 4.4: Number of daily generated alerts together with the daily air temperature and precipitation during summer 2022 in Gothenburg.

in Figure 4.4. As previously mentioned, the number of generated alerts during summer was much lower compared to alerts generated during winter and spring. High precipitation levels and air temperatures above 0 $°C$ increase the risk of hydroplaning which could be an explanation for the increase of generated alerts on the 25th of July and 4th of August. Most of the generated alerts during days with no or a small amount of precipitation are probably false positive alerts. There are many explanations for these false positives including hard breaking, rapid acceleration, or driving on speed bumps. To investigate this further, the locations of generated alerts during summer 2022 are visualized in Figure 4.5. The selected area shown in this figure is where the majority of the alerts were generated in Gothenburg during the summer of 2022.

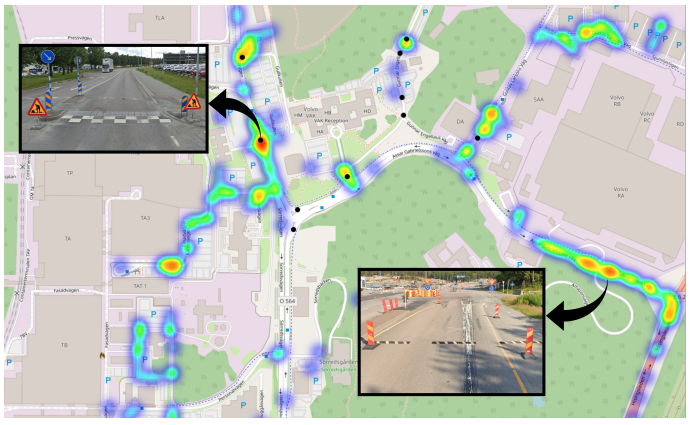

Figure 4.5: Heatmap of generated slippery road alerts within an area in Gothenburg during summer 2022.

The vast majority of generated alerts marked in red are close to speed bumps in this specific area. The reason why some of the false positive alerts are generated close to speed bumps could be due to hard braking or driving on the speed bump since both cause the estimated friction to change. A large number of alerts were generated close to a specific speed bump shown in the upper left corner. The location of this speed bump was included in the data from Trafikverket, thus it could be visualized as a black dot. However, speed bumps that were temporally or recently placed such as the speed bumps shown in the bottom right corner, were not included in the road data and could therefore not be used for further analysis and processing.

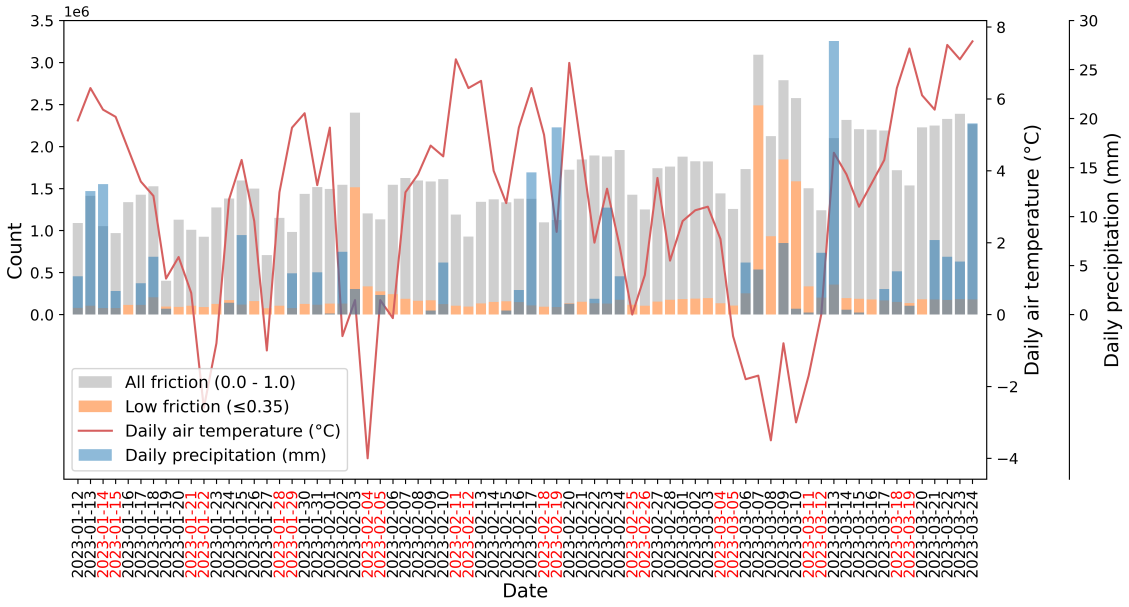

Figure 4.6: Number of friction measurements against the daily air temperature and daily precipitation levels.

As can be seen in Figure 4.6, both the daily air temperature and precipitation from SMHI correlate with the number of low estimated friction coefficients sent to the cloud each day. The number of low friction values is drastically increased only when the daily air temperature is below 0 °C and the daily precipitation is above 0 mm. These days were the 3rd of February and the 7th to 10th of March. This is reflected in the number of generated alerts previously shown in Figure 4.3. For other days, even though the daily precipitation was above 0 mm, the number of low friction values sent to the cloud was not increased much since the daily air temperature was above 0 °C. Hydroplaning might be an explanation for some of the true positives for these cases. Furthermore, some low friction values are sent to the cloud even though the daily air temperature was above 0 °C and the daily precipitation was 0 mm. These data points most likely correspond to false positive situations.

5

Methods

This chapter describes the methods used in this thesis and how the project was carried out based on the data analysis from the previous chapter. Section 5.1 begins with defining true- and false positives in the context of slippery road alerts. In Section 5.2, the proposed solution, a false positive filter is presented. Section 5.3 describes how the raw vehicle data was processed including sequence generation, sequence mapping, and sequence grouping. Section 5.4 describes the final data sets used to train, test and evaluate the machine learning models for each approach. Section 5.5 describes the architectures of the machine learning models and the experiments conducted in this thesis. Finally, Section 5.6 describes the evaluation process of the final models for both the autoencoder- and active learning approach.

5.1 Defining True- and False Positive SRAs

The first step of the project was to define true- and false positives in the context of slippery road alerts (SRAs). There are many possible causes for SRAs; some of them are ambiguous and therefore more challenging to classify. One can for example argue that SRAs that are generated when vehicles are driven on a gravel road should be classified as a true positive alert since it caused the vehicle to slip. However, since alerts caused by these types of situations depend on many other parameters and rarely occur in the same location, the alert might not be useful for the driver. Ultimately, defining true- and false positive SRAs can be divided into two types of definitions, one where the actual slipperiness is central and the other that focuses more on the value of receiving the alert. For this project, true positive alerts are defined as alerts generated in situations that lead to loss of traction. Conversely, false positive alerts are defined as alerts generated in situations when the vehicle did not slip.

Table 5.1: Definition of true- and false positive slippery road alerts.

	True positives	False positives
1	Ice/snow slip	Driving on speed bump
2	Hydroplaning	Driving on road crack/pothole
3	Gravel slip	Making a sharp turn
4	Mud slip	Rapid braking
5	Slip on wet leaves	High acceleration

Some of the most common causes of true- and false-positive alerts using this definition are shown in Table 3.1. In this definition, true positive SRAs depend on road type and environmental factors. Conversely, false positive SRAs are more related to road quality, road design, and vehicle maneuvering. The available data for this thesis sets the limit for which of the true- and false-positive alerts can be considered in developing the false-positive filter. Hence, false positive causes 2-5 could not directly be classified since the available data lacks features that are needed to classify them.

5.2 False Positive Filter

The main goal of this thesis was to create a false positive (FP) filter capable of reducing the number of generated false positive alerts in a (SRA) system as shown in Figure 5.1. Filtering solely on the current weather conditions for each specific situation is not enough for a rigorous solution. For example, some situations when the probability of ice formation on the road surface is high can still be caused by false positive situations e.g. driving on a speed bump or a pothole.

The idea was to create sequences of the provided vehicle data, train ML models on that data, and finally remove sequences corresponding to false positive situations e.g. driving on a speed bump or making a sharp turn. The sequential data corresponding to true positive situations is then passed to an SRA system where it is processed to create SRAs that are sent to other nearby vehicles.

There are two main reasons why we focused on creating a false positive filter. First, implementing an entire SRA system was impractical due to the limited time of the project. A false positive filter that acts alongside a current SRA system is less time-consuming to implement. Second, the bursts of data points are even further processed and filtered in current SRA systems. Hence, the number of valuable features for training the machine learning models is reduced going further into the system. Therefore, the filter was placed as close to the raw vehicle data as possible, i.e. when the data is sent to the cloud.

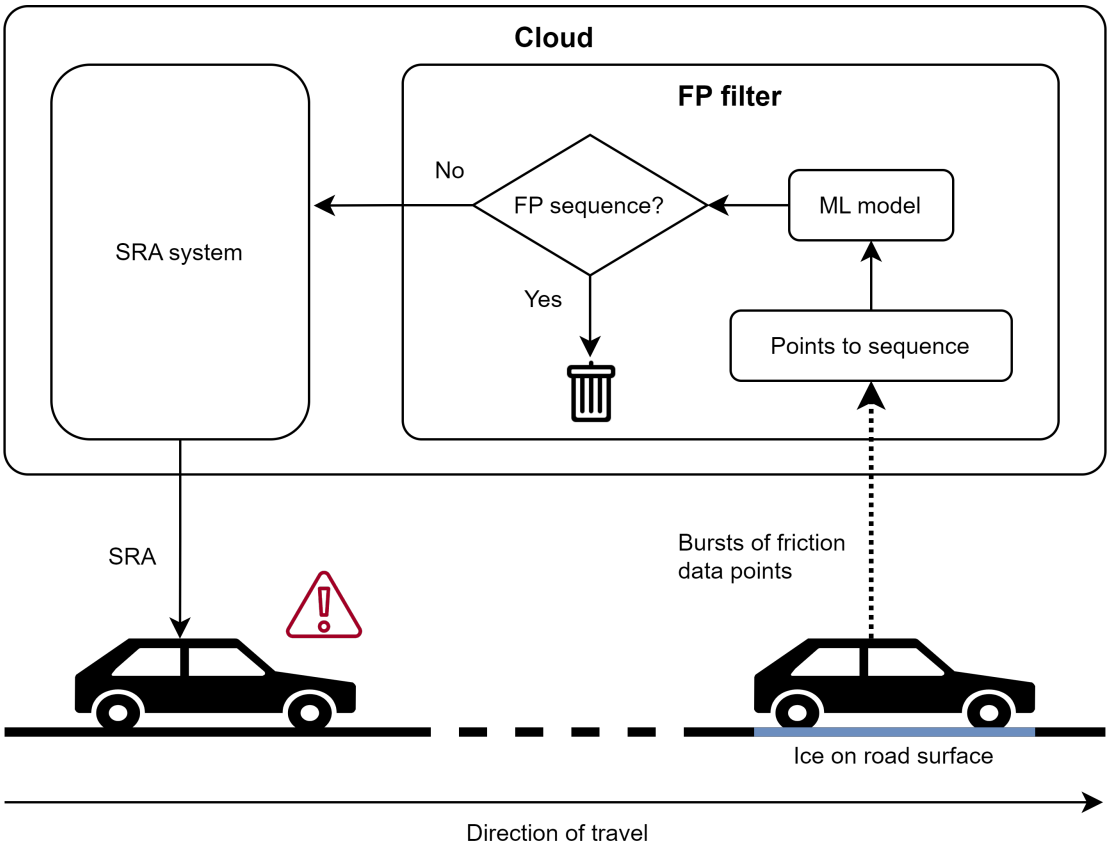

Figure 5.1: High-level overview of the proposed false positive filter.

5.3 Data Preprocessing

The process of generating a data set from the raw vehicle data is divided into the following three main parts: sequence generation, sequence mapping, and sequence grouping. First, the unprocessed vehicle data was clustered based on both time and location to form sequences for unique vehicles. Second, each sequence was mapped to weather stations, speed bump locations, and road types. Finally, the sequential vehicle data was grouped to create the final data sets used to train, test, and evaluate the machine learning models.

5.3.1 Sequence Generation

The available vehicle data consisted of unlabeled spatial-temporal data points including the estimated friction coefficient, friction quality, temperature, and wiper speed. Based on the initial analysis of this data, it seemed difficult to distinguish between false- from true-positives solely on point measurements with the available features. A single point measurement consisting of the friction coefficient and friction quality is not informative enough to determine if the point corresponds to a true- or false-positive situation. There are many other variables needed to make accurate predictions of the situation including vehicle speed and forces acting on the wheels. Additionally, the available data was unlabeled which increases the complexity of the

task of detecting and reducing false positive alerts.

The data analysis revealed that vehicles generate sequences of data points. A hypothesis was that both the estimated friction coefficient and friction quality will change over time differently depending on if the situation corresponds to a true positive situation e.g. driving on ice or a false positive situation e.g. driving on a speed bump. Furthermore, creating sequences of the vehicle data introduces dependencies between data points in consecutive order. This makes sequences more informative than a single point measurement. The idea was therefore to train machine learning models to learn patterns in the sequential vehicle data such that sequences corresponding to false positive situations could be removed. The process of creating these sequences was based on both temporal and spatial clustering.

The temporal clustering method is shown in Figure 5.2. First, each data point was grouped by the vehicle identification number (VIN) and sorted by the creation time. The creation time represents when each specific data point was created in the cloud and was the only accessible time variable in the vehicle data. The main goal of the temporal clustering was to form rough initial clusters that would reduce the execution time of the spatial clustering later on.

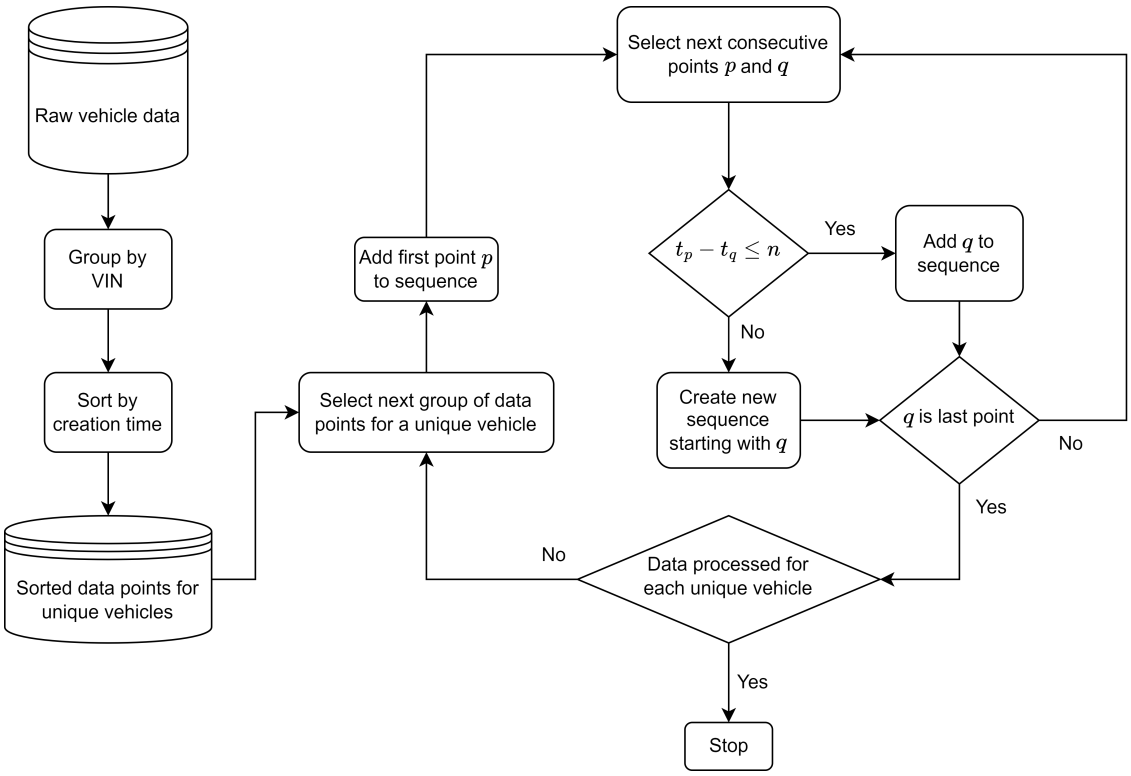

Figure 5.2: Temporal clustering of the raw vehicle data.

For a specific vehicle, two consecutive points p and q are considered to belong to the same sequence if the creation time between the points t_p and t_q is within n seconds. Otherwise, the points are placed into separate sequences. A too-small value for n would result in a split of sequences belonging to the same situation. Conversely, a too-large value for n could result in spatial collisions of sequences for a specific

vehicle. For example, a vehicle might generate a sequence of points at time t_0 on location l and might generate a sequence at time t_1 on the same location l. Hence, the spatial clustering later on would cluster these sequences into one single sequence which should be avoided since they do not correspond to the same situation. For this initial clustering, n was set to 10 seconds to ensure that each point for a specific situation is captured while also minimizing the risk of spatial collisions of sequences. After adding the next consecutive point q to the current sequence or creating a new sequence starting with q, the next consecutive pair of data points will consist of the point q and the next point in the list of points sorted by the creation times. The loop continues until each data point for each vehicle belongs to a sequence.

The sequences generated from the initial temporal clustering were further clustered using the DBSCAN clustering algorithm with Haversine distance as the distance measure, which is covered in section 3.4. The value of epsilon in DBSCAN should be as small as possible such that points that are related to the same situation are clustered together. However, a too-small epsilon would split sequences that relate to the same situation into different clusters. Conversely, a too-large value for epsilon would have the opposite effect, i.e. larger clusters containing sequences from multiple situations. Creating sequences of the vehicle data proved to be a challenging task since many unknown variables such as vehicle speed, road surface, and road construction affect the properties of a sequence including its length, sparseness, and form (e.g. straight or curved).

During the analysis of the vehicle data, it was discovered that data points on highways were more sparse compared to data points on other roads. Using a fixed small value for epsilon in DBSCAN would lead to multiple clusters for data points that relate to the same situation on highways. Increasing epsilon would solve this issue on highways. However, it would introduce issues on other roads where points were more densely packed. Instead, the value for epsilon was computed based on the road maximum speed (RMS) in km/h for each data point, as follows:

$$\epsilon = \left(\frac{RMS}{3.6}\right) k \tag{5.1}$$

The road maximum speed was divided by 3.6 to go from km/h to m/s and multiplied with a constant k that was tuned visually and finally set to 1. For example, if the vehicle was driven on a road with a maximum road speed of 80 km/h, the value for epsilon would be set to 22 meters. Hence, for two points to be considered neighbors in DBSCAN, the distance between the points would need to be less than or equal to 22 meters when driving on a road with a maximum legal speed of 80 km/h. Ultimately, after running DBSCAN on the entire vehicle data, a total of 6 million sequences were generated.

5.3.2 Sequence Mapping

After the sequence generation process of the vehicle data, each sequence was mapped to weather stations, speed bump locations, and road types. The mapping was implemented using spatial hashing (described in Section 3.6) which significantly reduced

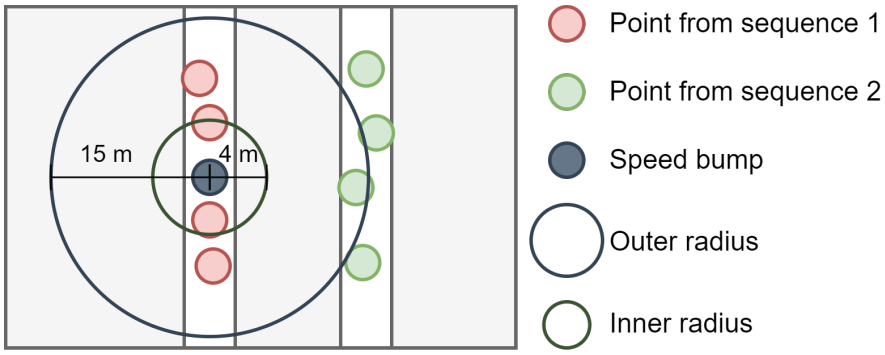

Figure 5.3: Sequence to speed bump matching.

the number of comparisons needed. The gathered vehicle data contained approximately 115 million data points which would result in billions of comparisons needed using a naive approach.

Weather station mapping

Each sequence was mapped to the closest weather station such that the weather data would more accurately represent the current weather conditions for specific sequences. However, it was impractical and unnecessary to compare the distances between every point within a sequence with the locations of the weather stations. Instead, a single point from a sequence was sampled to represent the location of the entire sequence. The distance between this point and the locations of each weather station was computed. The weather station closest to the point was then mapped to the sequence to which the point belonged. This procedure was applied for each sequence in the sequential vehicle data. Finally, the current temperature and precipitation levels were mapped to the closest half-hour of the sequential vehicle data since the weather data had a time interval of 30 minutes.

Speed bump location mapping

The mapping of speed bump locations was implemented in two steps. First, a sequence was considered to be close to a speed bump if each point of the sequence was within a radius of 15 meters of the speed bump, as shown in Figure 5.3. Points outside this radius were removed from the sequence since they were assumed to not relate to the same situation. However, this could lead to wrong mappings, e.g. sequences being mapped to speed bumps on a different road. Second, to counteract this issue, at least one point of a sequence needed to be within a radius of 4 meters from the speed bump.

Road type mapping

The mapping of sequences to road types was more complicated than the other mappings since the road type data contained polygons in contrast to singular points for the weather stations and speed bump locations. The road type data contained data for asphalt and gravel roads. Hence, it was only necessary to map points within each sequence to gravel roads since the rest of the sequences could be assumed to belong to asphalt roads. To map the sequences to gravel roads, the polygons were first converted to a set of coordinates. Points within a sequence were then mapped to a gravel road if any point was within a radius of 30 meters of a gravel road point.

5.3.3 Sequence Grouping

The final data sets used to train, test, and evaluate the machine learning models were created by grouping the sequential vehicle data on the current weather conditions, speed bump locations, and road types.

The sequential vehicle data was grouped based on the current weather conditions from Trafikverket using the flowchart shown in Figure 5.4. The flowchart was created based on assumptions under which weather conditions black ice and hydroplaning are likely to occur.

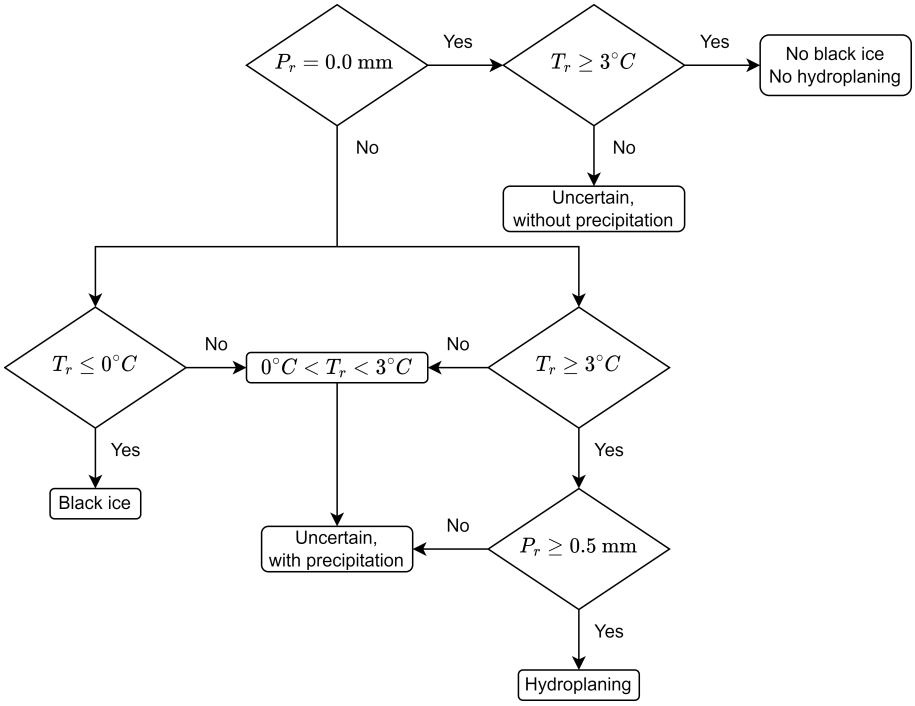

Figure 5.4: Assumed outcomes of different weather conditions.

For example, the probability of ice formation on the road surface is increased when the precipitation level near the road surface P_r is above 0 millimeters and the road temperature T_r is below or equal to 0 $°C$. However, the precipitation levels may have been above 0 millimeters a few hours before the sequence was created. This was taken into account by computing the mean precipitation of the previous three hours. Furthermore, the risk of hydroplaning was assumed to increase when the precipitation levels are above or equal to 0.5 millimeters and the road temperature is above or equal to 3 $°C$. In some weather conditions such as when the precipitation levels are above 0 millimeters and the temperature is between 0 and 3 $°C$, it was assumed to be too uncertain if the risk of ice formation or hydroplaning is increased. After all, these measurements do not represent the true weather conditions for each sequence and they might deviate from the reality. In cases when there was no precipitation and the road temperature was above or equal to 3 $°C$, it was assumed to be no risk of either black ice or hydroplaning. However, ice can be formed on the road surface if the percentage of humidity is high on the road surface and the road temperature is low. Humidity was not taken into account in this thesis but could

potentially be used for more rigorous assumptions of when ice is formed on the road surface.

5.4 Final Data Sets

The final data sets contained sequences generated by unique vehicles from the vehicle data. Since the autoencoder- and active learning approaches are based on different techniques, two data sets were created in total; one for each approach.

5.4.1 Autoencoder Data Set

The data set for the autoencoder was grouped into the following three classes based on weather conditions, speed bump locations, and road type:

- Assumed true positive sequences on ice (ATP-ICE)
- Assumed false positive sequences on dry days (AFP-DRY)
- Assumed false positive sequences on speed bumps (AFP-SB)

Table 5.2 shows how the classes were divided. Note, the classes were formed based on assumptions of when true- and false-positive situations might have occurred from 5.4. Therefore, there is a possibility that some sequences were assigned to the wrong class e.g. actual false positives being classified as assumed true positives.

The idea was to train the autoencoder on a subset of ATP-ICE sequences such that it learns the distribution of this data. The rest of the data was then used to evaluate the model. The number of sequences on gravel roads was small and was therefore not used for the autoencoder approach.

Table 5.2: Classes for the autoencoder data set.

	ATP-ICE	**AFP-DRY**	**AFP-SB**
Road temperature	$T_r \leq 0$	$T_r \geq 3$	$T_r \geq 3$
Road precipitation	$P_r > 0$	$P_r = 0$	$P_r = 0$
On speed bump	No	No	Yes
On gravel road	No	No	No

The ATP-ICE class was split into train-, validation-, and test-data sets containing 80%, 10%, and 10% of the entire ATP-ICE class respectively. Hence, the train ATP-ICE data set was used to train the autoencoders while the rest of the sequences were used for validation during training and testing after training. In addition to the test ATP-ICE data set, each sequence in the other assumed false positive classes AFP-DRY and AFP-SB were also used to test and evaluate the trained autoencoders.

Since false positive alerts occur when the friction coefficients are below or equal to 0.35, sequences that only contained friction coefficients above 0.35 were removed. Hence, at least one point within each sequence needed to have a friction coefficient below or equal to 0.35.

5.4.2 Active Learning Data Set

For the active learning approach, two separate data sets were prepared from the sequential vehicle data. First, sequences were grouped using weather and road features to determine which sequences corresponded to true and false positives. This was used to create pseudo labels for the data set used in this approach. The main features used for clustering were road temperature, precipitation, and road obstacles such as speed bumps and raised crosswalks. This clustering was based on the data analysis and literature studies to create a pseudo-labeled data set.

After this initial clustering, there was a large discrepancy between the size of the true- and false-positive classes, where true positives were significantly more in number. Therefore, these sequences were also grouped based on locations and by creating groups with sequences close to the weather stations as described in Section 5.3. Since these stations are mostly located in more rural areas, the most central parts of Gothenburg were also used to get more varied true positive sequences. This resulted in the following classes,

- Assumed true positive sequences near weather stations on ice (ICE-STAT)
- Assumed true positive sequences in central Gothenburg on ice (ICE-METRO)
- Assumed false positive sequences on dry days (SB-DRY)
- Assumed false positive sequences on raised crosswalks (RCW-DRY)

Additionally, Table 5.3 shows how the classes were divided by weather- and road conditions.

Table 5.3: Certain classes for active learning data set.

	ICE-STAT	**ICE-METRO**	**SB-DRY**	**RCW-DRY**
Road temperature	$T_r \leq 0$	$T_r \leq 0$	$T_r \geq 3$	$T_r \geq 3$
Road precipitation	$P_r > 0$	$P_r > 0$	$P_r = 0$	$P_r = 0$
On road obstacle	No	No	Yes	Yes
On gravel road	No	No	No	No
Labeling	ATP	ATP	AFP	AFP

A common problem with active learning is the reinforcement of existing bias in the initial data set. The classes were grouped based on weather conditions and road features leading to a data set that was highly biased on the temperature and precipitation. To counteract this, additional classes were defined that were less dependent on both temperature and perception. The additional true positive classes were gravel road sequences from dry days and sequences during heavy rain and high temperature. The false positive (FP) classes were instead road-obstacle sequences but on either colder or wet conditions. These FP classes were created to reduce the bias of weather or temperature while still avoiding snow or ice conditions that are more probably of causing true positives. These additional classes are the following:

- Assumed true positive sequences near weather stations in heavy rain (HP-STAT)
- Assumed true positive sequences in central Gothenburg in heavy rain (HP-METRO)
- Assumed true positive sequences on dry days (ATP-GR)
- Assumed false positive sequences on raised crosswalks in colder temperatures (RCW-COLD)
- Assumed false positive sequences on raised crosswalks in light rain (RCW-WET)
- Assumed false positive sequences on speed bumps in colder temperatures (SB-COLD)
- Assumed false positive sequences on speed bumps in light rain (SB-WET)

Additionally, the clustering conditions for these classes are shown in Tables 5.4 and 5.5

Table 5.4: Uncertain true positive classes for active learning data set.

Class	**HP-STAT**	**HP-METRO**	**GR-DRY**
Road temperature	$T_r \geq 3$	$T_r \geq 3$	$T_r \geq 3$
Road precipitation	$P_r > 0.5$	$P_r > 0.5$	$P_r = 0$
On road obstacle	No	No	No
On gravel road	No	No	Yes
Labeling	ATP	ATP	ATP

Table 5.5: Uncertain false positive classes for active learning data set.

Class	**SB-COLD**	**RCW-COLD**	**SB-WET**	**RCW-WET**
Road temperature	$T_r \leq 0$	$T_r \leq 0$	$T_r \geq 3$	$T_r \geq 3$
Road precipitation	$P_r = 0$	$P_r = 0$	$P_r > 0$	$P_r > 0$
On road obstacle	Yes	Yes	Yes	Yes
On gravel road	No	No	No	No
Labeling	AFP	AFP	AFP	AFP

For validating the trained classifiers it is important that the validation data set has balanced classes, meaning their sizes are similar. An imbalanced data set with a majority of sequences from a single class could lead to misleading accuracy and f1-scores. To avoid this issue, each class was divided into four groups: ATP-certain, and ATP-uncertain. AFP-certain and AFP-uncertain were the maximum sizes of these groups decided by the size of the smallest group. Additionally, by using stratification when splitting the data into training and validation data sets, the class balance was kept within these data sets as well. After the labeled data set was defined, the remaining unlabeled sequences were used to create an unlabeled data set that was used to sample new data during the active learning iterations. Similarly to the data set used for the autoencoders, sequences with only high friction measurements were removed.

5.5 Machine Learning Methods

The available data for this project was unlabeled. Hence, only unsupervised and possibly semi-supervised machine learning techniques could be applied to filter out assumed false positive (AFP) slippery road alerts (SRAs). Numerous techniques for anomaly detection and false positive reduction for multivariate time series were reviewed; these methods are presented in Section 2. Ultimately, autoencoder and active learning seemed to be promising techniques for reducing false positive alerts in this project due to available resources and time. They were therefore used as the two main approaches for this thesis. These methods are described in Section 3.5.4 and 3.3.2 respectively. Each of the models developed in this study used the LSTM and GRU architectures which are further explained in Section 3.5.1.

5.5.1 Autoencoder

In this thesis, both GRU- and LSTM-based autoencoders were developed to investigate how these architectures affect the performance of the proposed false positive filter. LSTM networks can learn long-term dependencies well but tend to be slow in both training and execution due to the complexity of the network. In contrast, GRU networks have proved to be more efficient in both training and execution [49]. However, for longer sequences, LSTM networks generally outperform GRU networks [50].

The autoencoders were trained to learn the representation of assumed true positive (ATP) sequences when the probability of ice formation on the road surfaces was assumed to be high, i.e. sequences from the ATP-ICE class. As previously mentioned, the autoencoder approach assumes that each class in the data set contains a majority of correctly assigned sequences. However, this might not be the case due to the assumptions made in the creation of the data set. Furthermore, the trained autoencoders were used to remove assumed false positive (AFP) sequences by setting a threshold on the reconstruction loss.

Both the GRU- and LSTM autoencoders consisted of two layers of their corresponding cell type for both the encoder- and decoder part of the architecture as shown in Figure 5.5.

The input $\boldsymbol{X}$ to the encoder is a tensor consisting of a batch of N sequences with length T as follows:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_{1,1}} & \dots & \boldsymbol{x_{1,T}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{x_{N,1}} & \dots & \boldsymbol{x_{N,T}} \end{bmatrix}$$

Additionally, each vector $\boldsymbol{x_{n,t}}$ contains values for any number of features F e.g. friction coefficient and friction quality. Hence, the shape of the input to the autoencoders was (N, T, F). Furthermore, the input tensor is encoded by the two layers in the encoder. Therefore the shape of the output tensor of the first layer is (N, T, H),

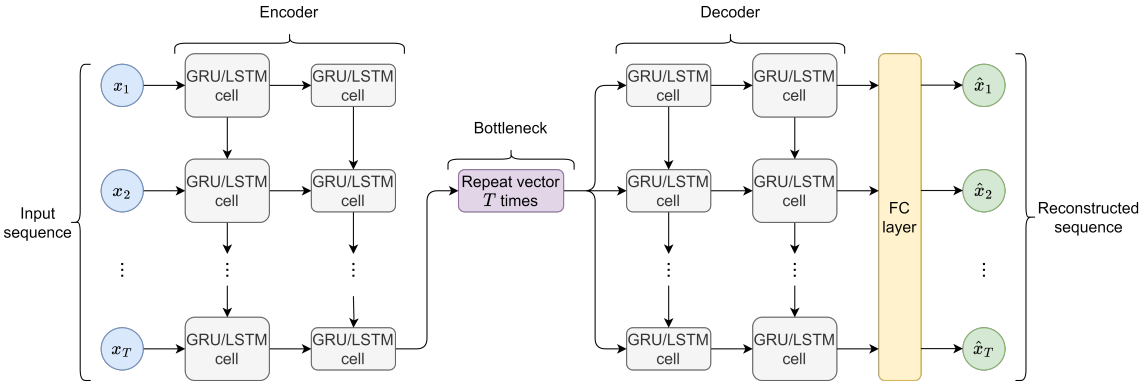

Figure 5.5: Structure of the GRU and LSTM autoencoders.

where H is the hidden size. This tensor is passed to the second layer in the encoder where it is processed to a latent vector representation of shape $(N, 1, \frac{H}{2})$. Hence, in the encoder, the hidden size of the first layer is twice as large compared to the hidden size in the second layer. The output of the last cell in the second layer in the encoder represents the compressed version of the input tensor. Since the goal of the autoencoder is to reconstruct the input, the single feature vector in the bottleneck is repeated T times along the second dimension. This means that the shape of the tensor is changed from $(N, 1, H)$ to (N, T, H). The repeated tensor is then passed to the first layer in the decoder where it is up-scaled into the same shape as the input tensor, i.e. (N, T, F). Finally, the decoded vector is passed to a fully connected (FC) layer to create the reconstructed output $\hat{\boldsymbol{X}}$:

$$\hat{\boldsymbol{X}} = \begin{bmatrix} \hat{\boldsymbol{x}}_{\boldsymbol{1,1}} & \dots & \hat{\boldsymbol{x}}_{\boldsymbol{1,T}} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{x}}_{\boldsymbol{N,T}} & \dots & \hat{\boldsymbol{x}}_{\boldsymbol{N,T}} \end{bmatrix}$$

The autoencoders expect that the input sequences are of equal length. Therefore, each sequence within a batch was padded by adding zeroes at the end of each sequence such that they matched the length of the longest sequence within the batch.

For the autoencoder approach, three experiments were conducted. In the first experiment, the goal was to train the autoencoders on different input data by alternating the sequence lengths, the number of features, and by including or excluding constant sequences. It was noted in the first experiment that the MSE loss was generally low which could result in numerical underflow. Therefore, for the second experiment, it was investigated if scaling the MSE loss improves the performance of the autoencoders and to assess if numeric underflow of the loss was an issue. The third experiment was dedicated to hyperparameter tuning using Optuna (further explained in Section 3.3.4) based on insights from the first experiment. The models were trained using either MSE or MAE as loss function in this experiment to investigate if different loss functions affect the autoencoder's ability to distinguish between true- and false positive sequences. Finally, the best trials from the third experi-

ment were evaluated by comparing the number of remaining true- and false-positive sequences after filtering.

The performance of autoencoders for anomaly detection is commonly evaluated by the use of metrics relying on ground truths. However, since the data in this thesis was completely unlabeled, these metrics could not be used. The evaluation should not only be based on the reconstruction loss of the autoencoder since a lower loss of the normal data does not necessarily indicate a better-performing autoencoder [46]. Furthermore, autoencoders can easily be trained to learn the identity function e.g. by using a too large embedding size. This results in an autoencoder that produces low reconstruction losses for any input and is thus unable to distinguish between normal and anomalous data [51].

There is currently a gap in the literature on how to evaluate autoencoders in unsupervised settings without ground truths. In this thesis, the evaluation of the autoencoders was therefore based on three objectives:

1. Low reconstruction loss for ATP class
2. Large difference between reconstruction loss of AFP- and AFP-classes
3. High proportion of ATP sequences remaining and high proportion of AFP sequences removed after filtering.

The trained models in each experiment were evaluated based on the first two objectives by plotting the distributions- and by computing the mean of the reconstruction loss for each class. The final evaluation of the best-performing autoencoders was evaluated based on the third objective by setting thresholds on the reconstruction loss and counting the number of sequences remaining after filtering.

AE - Experiment 1
The first experiment for the autoencoder approach was dedicated to testing different settings for the input data such as different sequence lengths, adding friction quality as a feature, and with or without constant sequences, as shown in Table 5.6. A hypothesis was that short sequences might be outliers caused by errors in the estimations or could potentially be artifacts from the sequence generation process. Too short sequences in the training data might lead to worse performance since they are less informative compared to longer sequences. Hence, the autoencoders might struggle to find patterns in the sequences when short sequences are included.

Low and high percentiles were used to remove short and long sequences respectively. Initially, sequences below the 10th percentile and above the 90th percentile were removed from each data set. With this setting, the shortest sequences contained 4 points while the longest contained 59 points. In experiment E1, the low percentile was increased to 50, i.e. the median of the sequence lengths. This resulted in the shortest sequences containing 18 points.

Both the GRU- and LSTM autoencoders were trained using the friction coefficient (FC) alone and together with the friction quality (FQ). Training the autoencoders with both features might help to distinguish between true- and false positive sequences.

Table 5.6: Settings for autoencoder experiment 1.

	Min length	Max length	Features	Only dynamic
1	4	59	FC	No
2	4	59	FC	Yes
3	18	59	FC	No
4	18	59	FC	Yes
5	4	59	FC and FQ	No
6	4	59	FC and FQ	Yes
7	18	59	FC and FQ	No
8	18	59	FC and FQ	Yes

During the analysis of the final data set, it was discovered that some sequences were constant, i.e. had constant friction coefficient values throughout the entire sequence. Similar to short sequences, constant sequences might be artifacts from the sequence generation procedure or might be caused by errors in the estimations of the friction coefficients. Including constant sequences can make it more difficult for the autoencoder to find patterns in the data. Therefore, in the first experiment, the models were trained on data that both included and excluded constant sequences to investigate how this affected the models' ability to distinguish between the assumed true- and false-positive sequences.

These settings were applied to both assumed true- and false-positive sequences. Otherwise, if the settings only would be applied to the training data, we would enforce bias such that the reconstruction loss would become larger for assumed false positive sequences. For example, if the model is trained on long sequences but the test data contains more short sequences, the reconstruction loss for the test data would be larger since the model has not been trained on short sequences, hence a bias on sequence length would be introduced.

For this initial experiment, the autoencoders were trained using the same hyperparameters shown in Table 5.7.

Table 5.7: Hyperparameter used in autoencoder experiment 1.

Hyperparameter	Value
Batch size	128
Optimizer	Adam
Learning rate	0.0001
Emb. dim.	64
Dropout	0.2
Epochs	100
Loss function	MSE

AE - Experiment 2

During the project, it was noted that the reconstruction losses produced by the autoencoders were generally low for any input sequence. Too low losses can result

in numeric underflow in the computation of the gradients during backpropagation and thus prevent the model from learning. Therefore, in the second experiment for the autoencoder approach, the LSTM autoencoder was trained by scaling the MSE loss by 100 such that numeric underflow of the reconstruction loss is prevented. The model was trained on setting 1 from the first experiment and using the hyperparameters shown in Table 5.8.

Table 5.8: Hyperparameters used in autoencoder experiment 2.

Hyperparameter	**Values**
Batch size	64
Optimizer	Adam
Learning rate	1e-4
Embedding size	16
Dropout	0.2

AE - Experiment 3

In the third experiment for the autoencoder approach, the main focus was to tune the hyperparameters of the optimal settings from the first experiment. Testing each combination of hyperparameters was not manageable since it would be too time-consuming. Instead, Optuna was used to tune the hyperparameters for 50 trials in which the models were trained for 20 epochs. Furthermore, the default Tree-structured Parzen Estimator (TPE) was used as the sampling method, and the objective was set to both minimize the average reconstruction loss and to maximize the difference between the average reconstruction loss of the ATP-ICE validation set and AFP-DRY set. As previously mentioned in Section 3.5.4, tuning the hyperparameters of the autoencoder by only minimizing the reconstruction loss is not enough since it will not necessarily result in the best-performing model in terms of its ability to distinguish between true- and false positive sequences. Finally, the hyperparameters tuned in this experiment are shown in Table 5.9.

Table 5.9: Hyperparameters tuned with Optuna for the autoencoders.

Hyperparameter	**Values**
Batch size	[16, 32, 64, 128]
Optimizer	[Adam, RMSprop, SGD]
Learning rate	[1e-2, 1e-3, 1e-4, 1e-5]
Embedding size	[4, 8, 16, 32, 64, 128]
Dropout	[0.0, 0.2, 0.5]

5.5.2 Active Learning

Active learning was used as an alternative approach to classify false positives. Unlike autoencoders that input unlabeled sequential data, the main idea of active learning is to label the data by training a classifier in an iterative process. Restrictions in time and resources prevented manual data gathering and labeling on a larger scale. By using active learning, only a small portion of labeled data is needed. This method uses this initial data set to train a model and then uses different sampling methods to select new data points from an unlabeled data set to improve the model further. This method consists of two parts, creating an initial model and then using active learning to improve it.

The result of the active learning approach is a classifier that directly can be used to remove false positive sequences. However, unlike the data for the autoencoder approach, additional features from the vehicle- and weather data were used for the classifiers. Most of these features did not change within the sequences and were therefore made into static variables of categorical and numeral types. Therefore, in addition to using GRU- and LSTM layers for the sequential data, fully connected layers, and embedding layers were used for the numerical and categorical features. Hence, the baseline classifiers used in this approach have different model architectures.

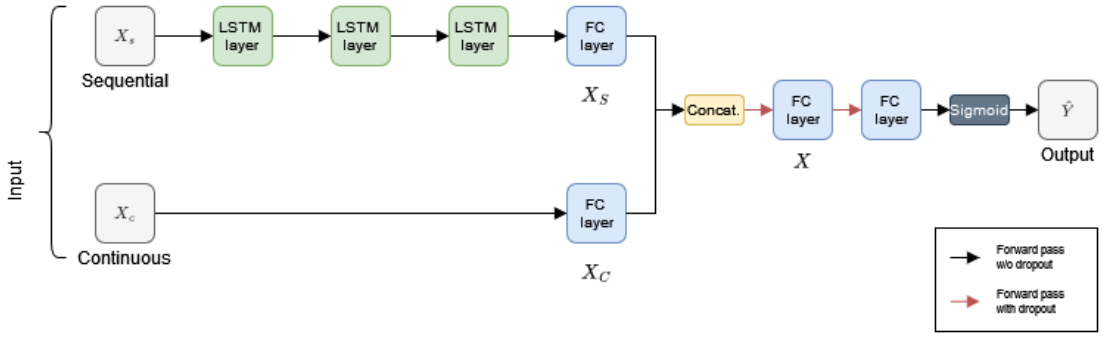

Figure 5.6: Baseline model for the active learning approach.

The baseline models using only the sequential data had a simpler architecture seen in Figure 5.6. The features used to train these models were the friction coefficient, friction quality, and sequence length. These features are passed through three LSTM layers resulting in the output X_s. The sequence length is instead passed through a fully connected layer resulting in the output X_c. X_s and X_c are then concatenated resulting in X. Finally, the concatenated output is passed through two linear layers and then to a sigmoid function. The output from this layer is the probability of the features belonging to assumed true- or false positive situations, represented as 0 or 1. The red arrows represent dropout layers that randomly sets neurons to zero to avoid overfitting.

The models that handled the additional vehicle- and weather features instead used the architecture seen in Figure 5.7. Unlike the baseline models, these models contained additional embedding layers to handle categorical variables. The output from

this layer was passed through a fully connected layer resulting in X_c. This output was then concatenated with the output from X_n before finally being combined with the sequential output X_1 resulting in X.

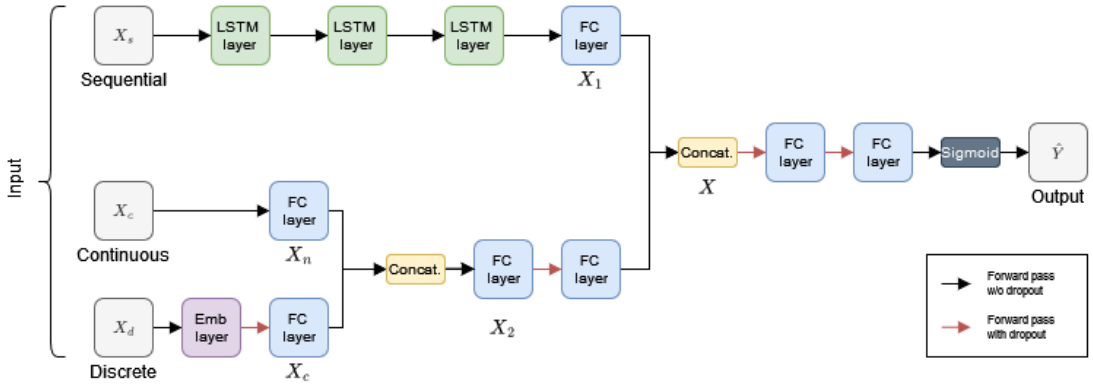

Figure 5.7: Final model for the active learning approach.

The initial classifiers were trained using hyperparameters based on initial experiments shown in Table 5.10. These hyperparameters were used in both experiments 1 and 2, whereas Optuna was used for experiment 3.

Table 5.10: Initial hyperparameters for the classifiers

Hyperparameter	Values
Batch size	128
Hidden Size	64
Learning rate	1e-5
Optimizer	Adam
Dropout Emb	0.3
Dropout FC	0.3

For experiments 1 and 2, a data set containing 1000 sequences from the certain classes and 500 sequences from the uncertain classes was used. However, in experiment 3, smaller subsets were tested to see if better performance could be achieved with a smaller starting data set. For all experiments, the same validation data set was used to keep them comparable.

AL - Experiment 1

The first experiment in the active learning approach was dedicated to training a classifier using only the sequential features. However, due to poor performance, the sequence length was added as an additional feature. The other settings that were tested in this experiment were model type, removing constant sequences, and finally removing short sequences. Sequences containing friction coefficients that did not change within the sequence were referred to as constant. The goal of this experiment was to see how different training data affect the performance of the baseline models. The tests from experiment 1 can be seen in Table 5.11.

Table 5.11: Settings for active learning experiment 1.

	Model type	Features	Only dynamic
1	LSTM	FC and FQ	No
2	GRU	FC and FQ	No
3	LSTM	FC and FQ	Yes
4	GRU	FC and FQ	Yes
5	LSTM	FC,FQ and Length	No
6	GRU	FC,FQ and Length	No
7	LSTM	FC,FQ and Length	Yes
8	GRU	FC,FQ and Length	Yes

Even when using sequence length as an additional feature and removing dynamic features, the performance was not sufficient. Therefore additional experiments using more features were made for the initial classifiers.

AL - Experiment 2
In the second experiment for the active learning approach, the performance of the models using different features was conducted. These experiments are shown in Table 5.12.

Table 5.12: Settings for active learning experiment 2.

	Model type	Features	Only dynamic
1	LSTM	Vehicle Features	No
2	GRU	Vehicle Features	No
3	LSTM	Vehicle Features	Yes
4	GRU	Vehicle Features	Yes
4	FNN	Weather Features	No
5	LSTM	All Features	No
6	GRU	All Features	No
7	LSTM	All Features	Yes
8	GRU	All Features	Yes

Using the insights from previous experiments, the best-performing model was selected and improved further using Optuna for hyperparameter tuning. The hyperparameters that were used for tuning and their ranges are shown in Table 5.13.

AL - Experiment 3
There are many tunable parameters for active learning including the size of the initial labeled data set, the number of samples to be added in each iteration, the sampling method, and when to terminate the iterations. Experiment three was therefore conducted to test these settings to achieve better performance.

The initial data used to train the models in this experiment consisted of the four previously mentioned groups: ATP-Certain, AFP-Certain, ATP-Uncertain, and AFP-Uncertain. The certain groups corresponded to 33% of the starting size each whereas

Table 5.13: Hyperparameters tuned with Optuna for the classifier model.

Hyperparameter	**Values**
Batch size	[16, 32, 64, 128]
Hidden Size	[32, 64, 128, 256]
Learning rate	[1e-2 - 1e-6]
Optimizer	[Adam, RMSprop, SGD]
Dropout Emb	[0.0 - 0.6]
Dropout FC	[0.0 - 0.6]

the uncertain groups corresponded to 17% each. Using a higher amount of uncertain classes lead to worse performance, whereas excluding them completely led to overfitting and biased models. The number of samples added per iteration was 10% of the start size meaning that after 10 iterations, the sizes doubled. To avoid adding samples that the models had trouble classifying a threshold was implemented based on the least confidence and margin scores. This was to avoid adding samples the model could not label accurately. The settings for experiment 3 are shown in Table 5.14

Table 5.14: Settings for active learning experiment 3.

	Start size	**Sampling method**
1	300	Uncertainty
2	300	Margin
3	600	Uncertainty
4	600	Margin
5	1500	Uncertainty
6	1500	Margin
7	3000	Uncertainty
8	3000	Margin

5.6 Evaluation

The available data was unlabeled meaning that common evaluation metrics relying on ground truths could not be used. This meant that a different approach for evaluation was needed. Ideally, the models should be evaluated alongside an SRA system since it would then be possible to compare the generated SRAs when using the models as filters and without them. However, this was not manageable due to the limited time of the project. Instead, the models were evaluated based on the number of remaining assumed true- and false-positive sequences after executing each model on a test data set. The test data set for the autoencoder contained 10% of all ATP-ICE sequences and all sequences from the AFP-DRY and AFP-SB sequences. For the classifiers, the test data contained 20% of all classes since the classifier was trained on sequences from each class, unlike the autoencoder. Importantly, the sequences in the test data set were not part of the training data used to train the models.

Furthermore, the data sets created in this project were created based on assumptions of when true- and false-positive sequences are likely to occur. It is difficult to determine if these assumptions were accurate. If the assumptions were accurate, then the amount of remaining true positives should be close to 100% while the amount of remaining false positives should be close to 0%. However, it is more likely that the assumptions made in the creation of the data sets are inaccurate. Therefore, a more realistic assumption is that 80-90% of the assumed true positives should ideally remain after filtering since some sequences were probably assigned to the wrong class.

The main idea of the autoencoder approach was to set a threshold on the reconstruction loss to differentiate between the true and false positives. Sequences receiving a reconstruction loss above the threshold are assumed to be false positive sequences and are therefore removed. Deciding the threshold involves a trade-off between the number of removed assumed true- and false positive sequences. As previously mentioned, since the classes were created based on assumptions, sequences could have been assigned to the wrong class. Therefore, two different thresholds were set for the reconstruction loss. The first threshold was set based on the 98th percentile of reconstruction loss for the ATP-ICE test set. The second threshold was set to maximize the difference between the percentage of remaining true- and false-positive sequences as follows:

$$\text{Threshold} = \text{argmax}(\%\text{ATP} - \%\text{AFP}) \tag{5.2}$$

For the active learning approach, four models were evaluated in total. These were the best performing models which used all features, being a GRU and LSTM model respectively. The two remaining models were the weather feature model and one using only vehicle features. Unlike the autoencoder approach, these models were used directly to filter the sequences since the output was predictions instead of reconstructions.

6

Results

This chapter presents the results of both the autoencoder and active learning approach in two sections. Section 6.1 begins with presenting the results of the autoencoder approach followed by Section 6.2 where the results from the active learning approach are presented.

6.1 Autoencoder

In this section, the results from the experiments of the GRU- and LSTM autoencoders are presented. As described in Section 5.5.1, three experiments were conducted for the autoencoder approach. In the first experiment, the data was manipulated by alternating the sequence length, the number of features, and including or excluding constant sequences. In the second experiment, the MSE loss was scaled to investigate how this affects the learning and performance of the model. The third experiment was dedicated to hyperparameter tuning and using different loss functions (MSE or MAE). Finally, the best-performing autoencoders were evaluated using the methods mentioned in Section 5.6.

AE - Experiment 1
In the first experiment, both the GRU- and LSTM autoencoders were trained on sequences from the ATP-ICE class with different properties and using the same hyperparameters described in Section 5.5.1. The mean MSE loss was computed for the ATP-ICE, AFP-DRY, and AFP-SB classes. For the ATP-ICE class, the mean MSE loss of both the train- and test data sets are presented together with the difference of the mean MSE losses between the test data set and AFP data sets. Finally, the distributions of the reconstruction losses for each class and the most optimal setting are presented.

Table 6.1: Results from experiment 1 - GRU-autoencoder

	Mean MSE loss				Mean MSE loss Difference	
	ATP-ICE (Train)	ATP-ICE (Test)	AFP-DRY	AFP-SB	ATP-ICE (Test) vs. AFP-DRY	ATP-ICE (Test) vs. AFP-SB
1	**0.00028**	**0.00026**	**0.00108**	**0.00094**	**0.00082**	**0.00068**
2	0.00035	0.00036	0.00158	0.00153	0.00122	0.00118
3	0.00057	0.00062	0.00242	0.00149	0.0018	0.00087
4	**0.00073**	**0.00069**	**0.00386**	**0.00253**	**0.00317**	**0.00184**
5	0.00211	0.00210	0.00157	0.00179	0.00053	0.00030
6	0.00311	0.00313	0.00283	0.00314	0.00030	0.00001
7	0.00486	0.00485	0.00362	0.00329	0.00123	0.00156
8	0.00619	0.00619	0.00552	0.00491	0.00067	0.00128

Table 6.2: Results from experiment 1 - LSTM-autoencoder

	Mean MSE loss				Mean MSE loss Difference	
	ATP-ICE (Train)	ATP-ICE (Test)	AFP-DRY	AFP-SB	ATP-ICE (Test) vs. AFP-DRY	ATP-ICE (Test) vs. AFP-SB
1	**0.00029**	**0.00029**	**0.00134**	**0.00108**	**0.00105**	**0.00080**
2	0.00032	0.00033	0.00158	0.00149	0.00125	0.00116
3	0.00084	0.00090	0.00376	0.00265	0.00286	0.00175
4	**0.00073**	**0.00071**	**0.00477**	**0.00317**	**0.00406**	**0.00246**
5	0.00298	0.00298	0.00237	0.00253	0.00061	0.00045
6	0.00411	0.00412	0.00371	0.00405	0.00042	0.00007
7	0.00540	0.00535	0.00453	0.00391	0.00082	0.00145
8	0.00758	0.00760	0.00770	0.00673	0.00010	0.00087

In general, using only the friction coefficient as a feature (settings 1 to 4) resulted in lower mean MSE loss for the assumed true positive sequences (ATP-ICE) as shown in Table 6.1 and Table 6.2. Setting 1 resulted in the lowest mean reconstruction loss for the ATP-ICE class. However, the largest mean MSE loss difference between the assumed true- and false-positive sequences was achieved in setting 4. Furthermore, the mean MSE loss difference for setting 4 was higher using the LSTM autoencoder. The LSTM autoencoder performed generally better in terms of lower mean reconstruction loss for the ATP-ICE class and higher mean MSE difference between the ATP-ICE and AFP classes. For settings 5-8, corresponding to when friction quality was included as an additional feature, the mean MSE loss was increased and the differences were decreased in general. Additionally, for settings 1-4, when the MSE loss of the assumed true positives is decreased, the MSE loss of the assumed false positives is also decreased. Furthermore, a lower mean MSE loss for the train- and test sets do not necessarily mean a high difference between the mean MSE loss of the assumed true-and false positives.

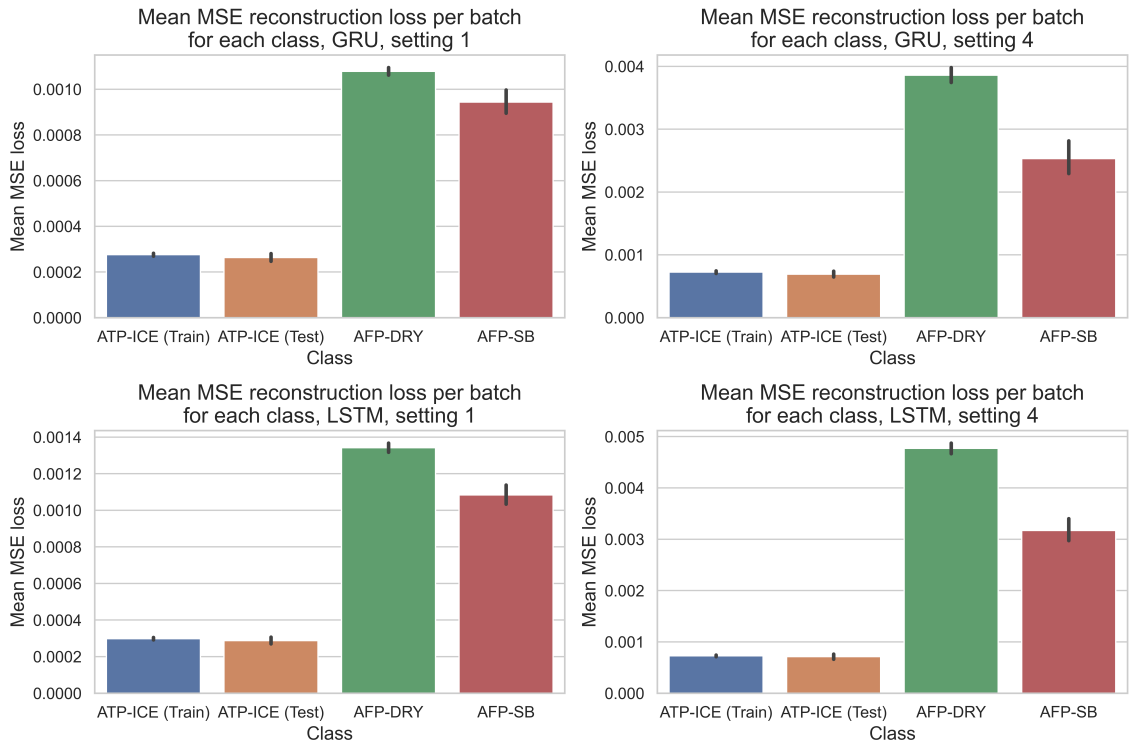

Figure 6.1: Mean MSE reconstruction losses of each class using the GRU- and LSTM autoencoders for setting 1 and 4.

In Figure 6.1 it is even more clear that the mean reconstruction loss for the assumed true positive sequences is lower compared to the mean reconstruction loss of the assumed false positive sequences. As previously mentioned, the difference is slightly larger for setting 4 compared to setting 1. There are overall small differences between GRU- and LSTM. The mean reconstruction losses for the speed bump sequences (AFP-SB) were lower compared to the mean reconstruction losses for the sequences on dry days (AFP-DRY).

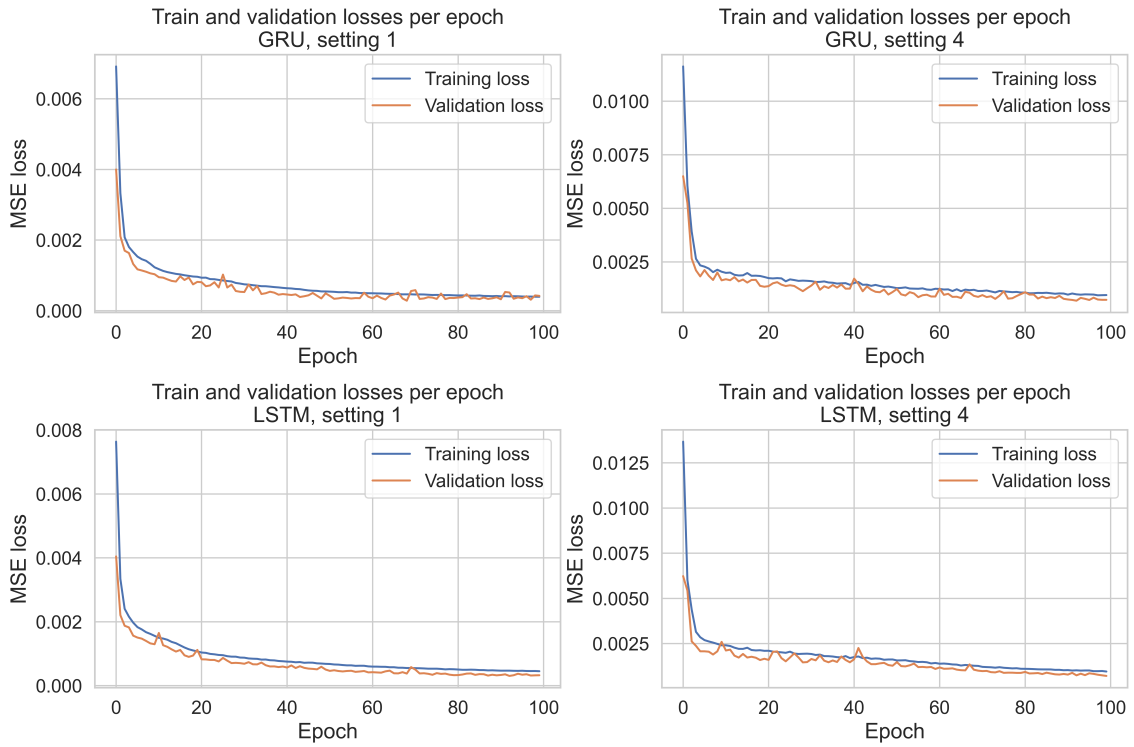

Figure 6.2: Train- and validation MSE losses for settings 1 and 4 using GRU- and LSTM autoencoder.

As shown in Figure 6.2, the curves of the train- and validation MSE losses for each epoch are similar. The validation loss fluctuates more compared to the training loss but ultimately converges to a similar loss value.

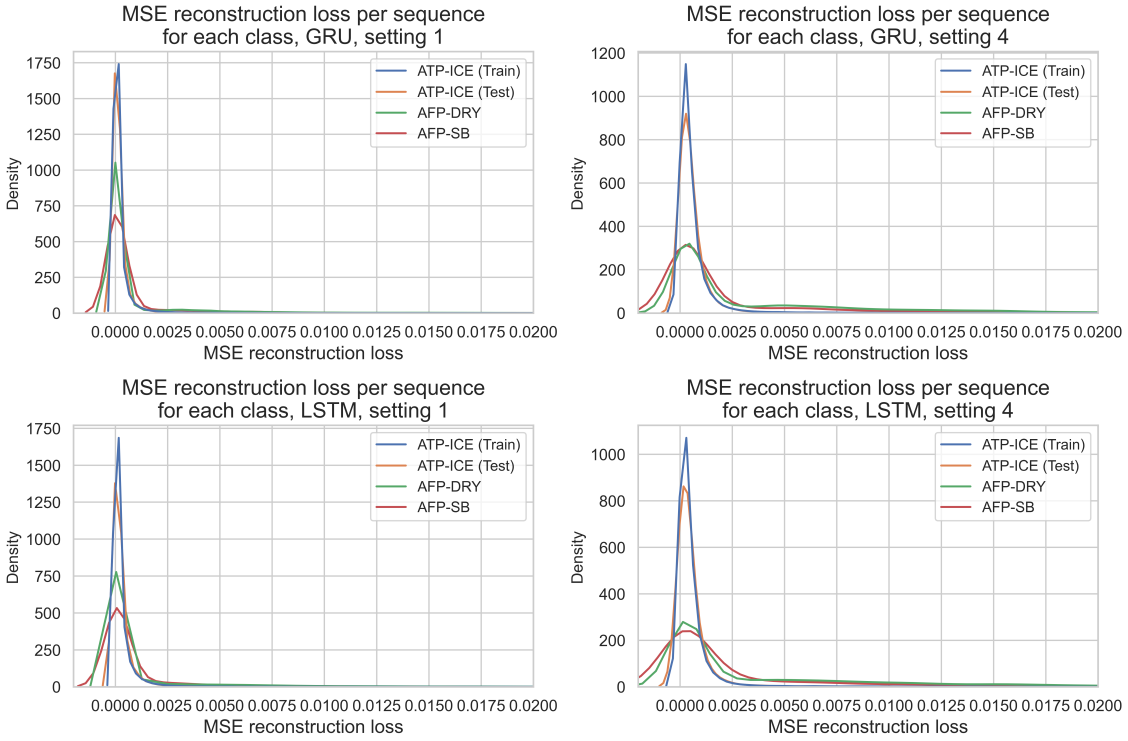

Figure 6.3: Distributions of reconstruction losses per sequence of each class using the GRU- and LSTM autoencoders for setting 1 and 4.

With single instance prediction, the distributions of the reconstruction losses for each class are almost identical as shown in Figure 6.3. There are small differences between GRU- and LSTM. The LSTM autoencoder produced slightly more separated distributions compared to the GRU autoencoder for setting 4.

AE - Experiment 2

Scaling the MSE loss did not result in any noticeable difference in the distributions of the reconstruction losses as shown in Figure 6.4.

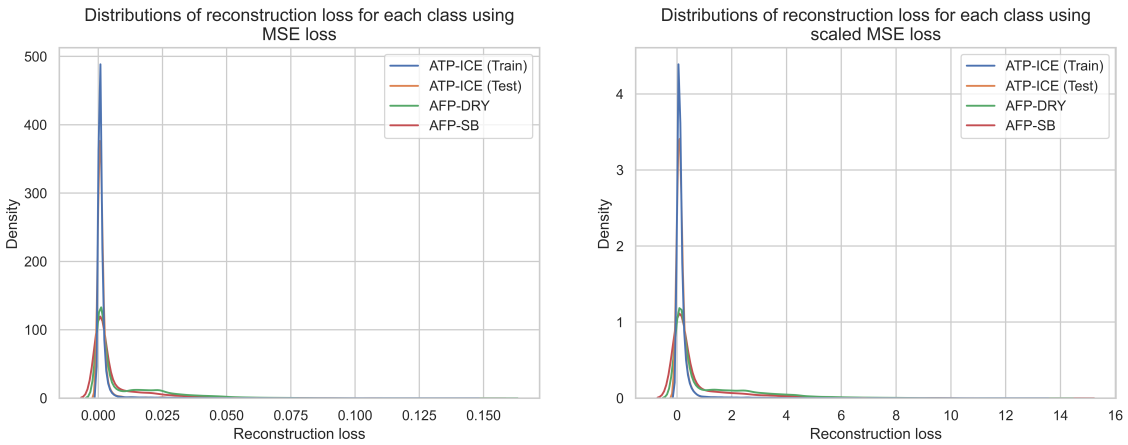

Figure 6.4: Distributions of the reconstruction losses for each class using MSE loss and scaled MSE loss.

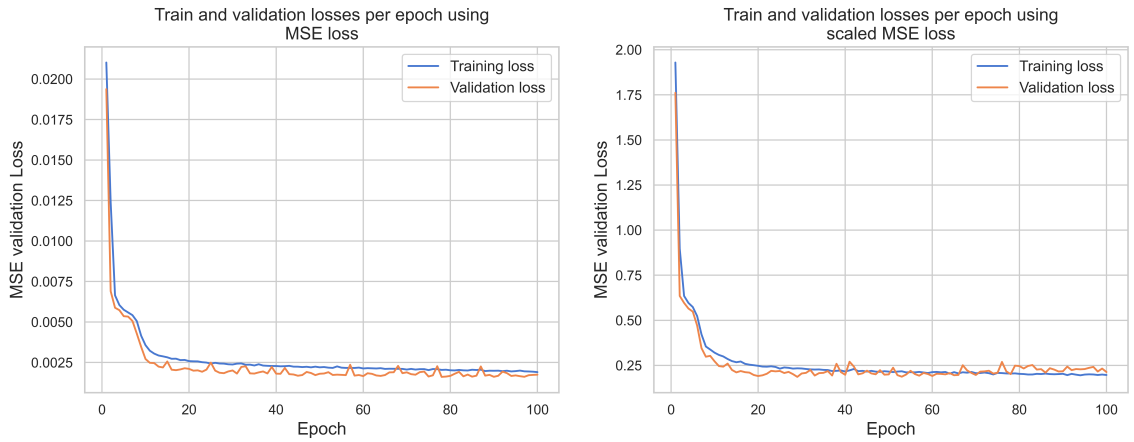

Figure 6.5: Train and validation loss for each epoch using regular MSE loss and scaled MSE loss.

The loss during training is similar as well when using regular MSE and scaled MSE as shown in Figure 6.5.

AE - Experiment 3

The third experiment for the autoencoder approach was dedicated to hyperparameter tuning using Optuna. From experiment 1, the difference in mean reconstruction loss between the assumed true- and false-positive sequences was larger in setting 4 with the LSTM autoencoder. Hence, setting 4 was used for hyperparameter tuning the LSTM autoencoder using both MAE and MSE as loss function. The results of each combination of autoencoder and loss function including Pareto front plots, hyperparameter importance, and hyperparameters are shown in Appendix A.

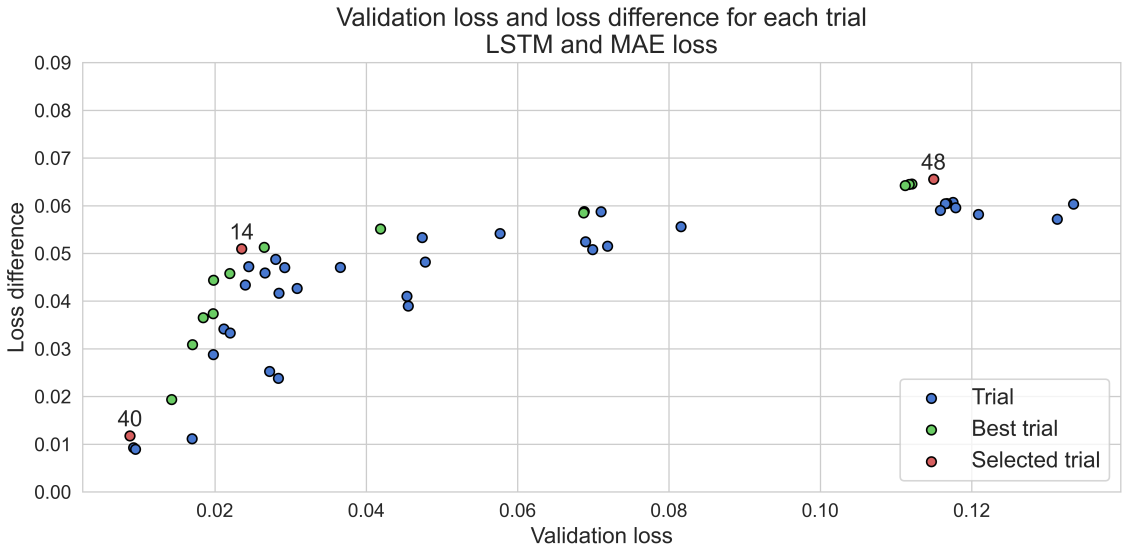

Figure 6.6: Validation loss and loss difference for each trial with the LSTM autoencoder using MAE as loss function.

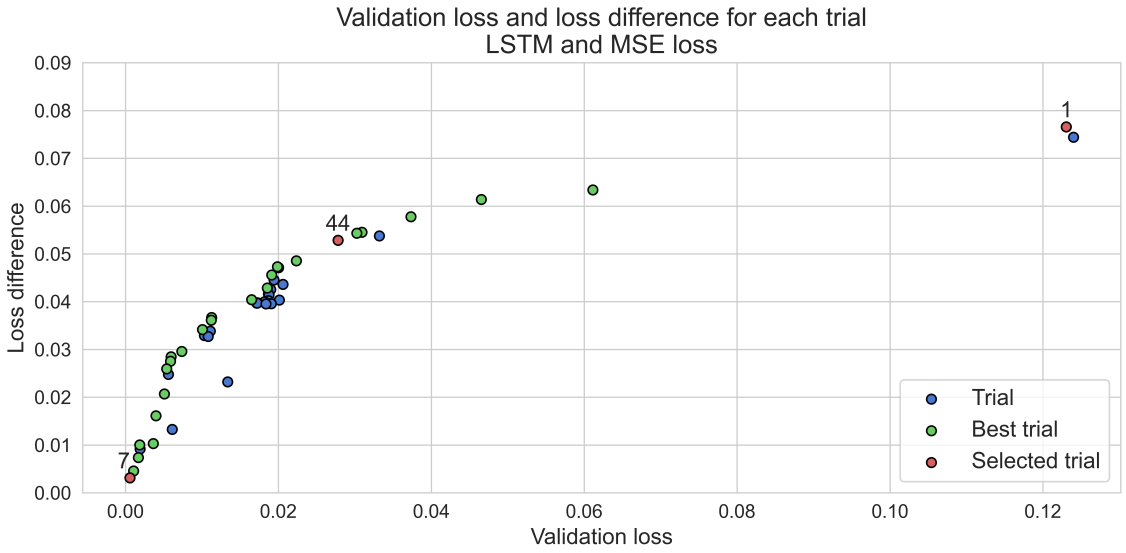

Figure 6.7: Validation loss and loss difference for each trial with the LSTM autoencoder using MSE as loss function.

Table 6.3: Autoencoder - Best hyperparameters

Loss function and Hyperparameters	**Selected trials**					
	40	14	48	7	44	1
Loss function	MAE	MAE	MAE	MSE	MSE	MSE
Batch size	32	16	16	32	32	64
Optimizer	Adam	Adam	SGD	Adam	SGD	SGD
Learning rate	0.0001	0.00001	0.0001	0.0001	0.00001	0.00001
Embedding size	128	32	64	128	16	4
Dropout	0.0	0.2	0.0	0.0	0.5	0.0

Figure 6.6 and Figure 6.7 shows the final validation loss and loss difference between the ATP-ICE validation set and AFP-DRY set for each trial using MAE and MSE as loss function respectively. The best trials (green and red) are trials in the Pareto front. Since the objectives were set to minimize the validation loss and maximize the loss difference, trials that are closer to the upper left corner are considered to be optimal. However, as can be seen in Figure 6.6 and 6.7, a trade-off between these objectives needs to be made when selecting the optimal trial since there is no single optimal trial. Therefore, three trials were selected (marked in red) for both the MAE- and MSE loss functions for further analysis.

The hyperparameters for each selected best trial from Optuna hyperparameter tuning are shown in Table 6.3. The LSTM autoencoder was trained for 100 epochs using the hyperparameters for each trial shown in this table. Furthermore, the distributions of the reconstruction loss for each trial are shown in Figure 6.8. Note, the loss values from MSE are in general lower compared to the loss values of MAE. Using a larger embedding size and Adam as the optimizer (trials 40 and 7) resulted in less separated distributions of the reconstruction loss. Conversely, the distribu-

tions of the reconstruction loss between the ATP-ICE and AFP sequences were more separated using a lower embedding size and SGD as the optimizer.

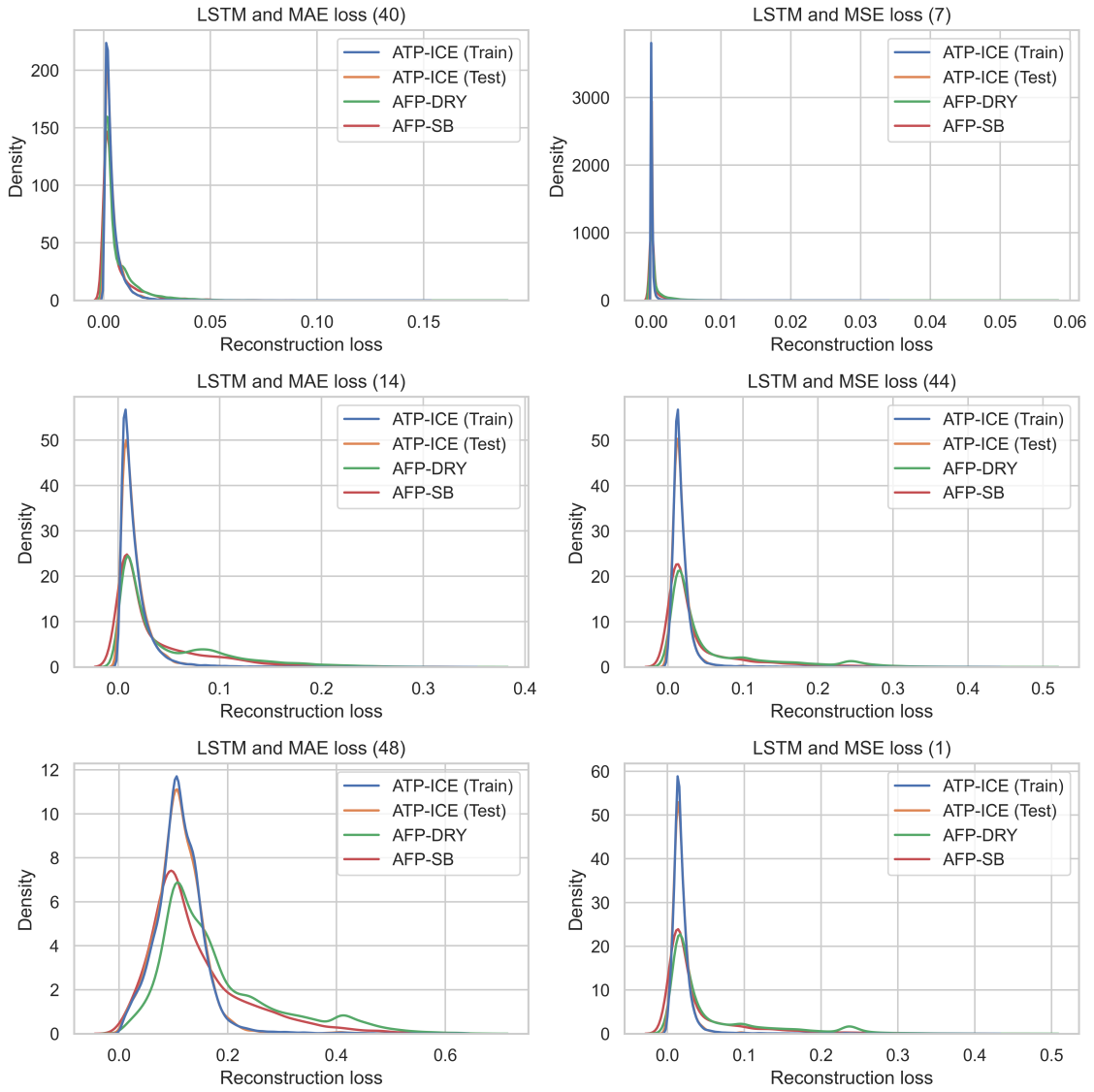

Figure 6.8: Distributions of reconstruction losses for each class and for each model trained with hyperparameters from the selected trials.

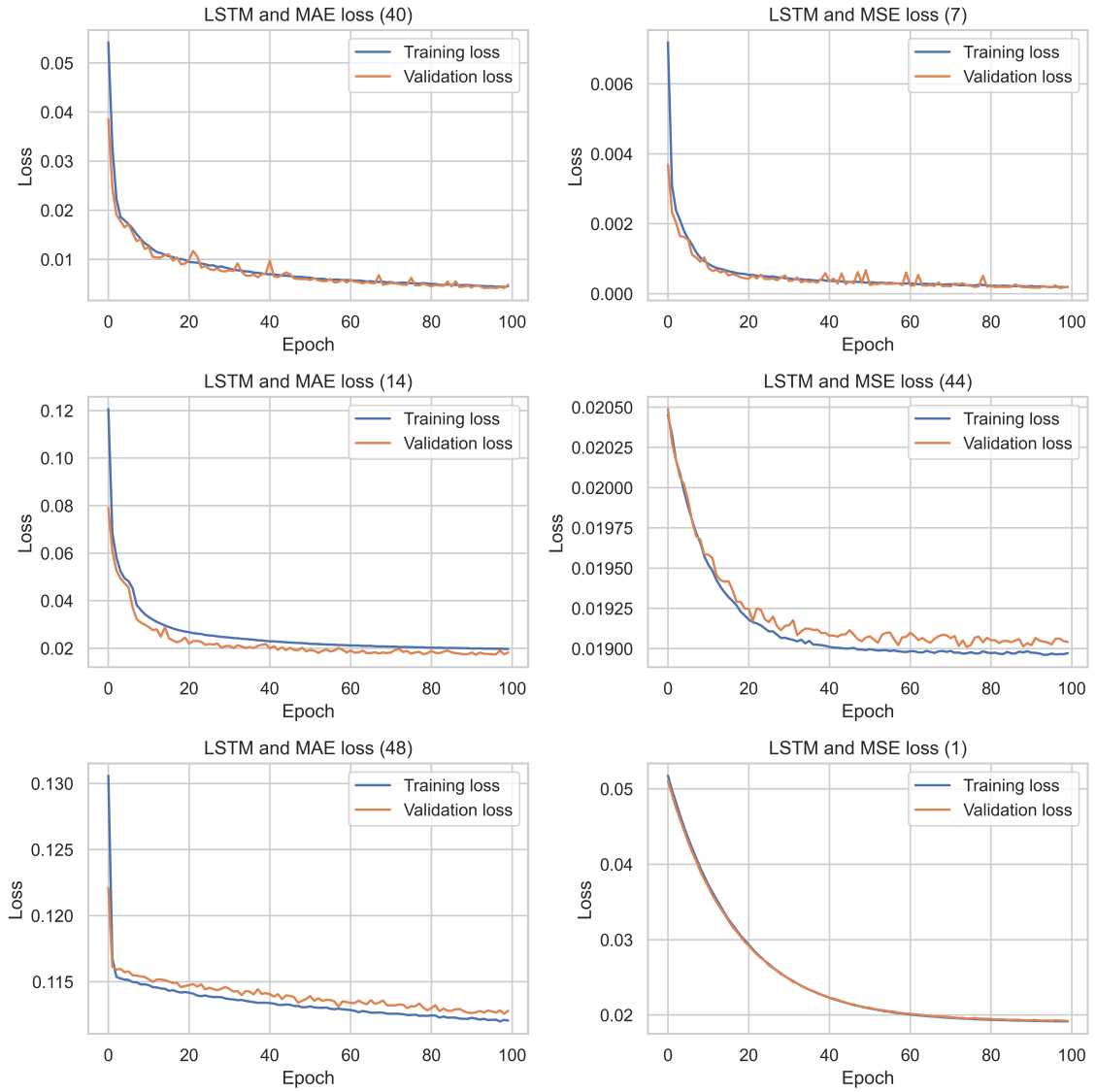

Figure 6.9: Train- and validation loss during training of models with hyperparameters from the selected trials.

As can be seen in Figure 6.9, the loss gradually decreased for each selected trial. The validation loss is slightly above the training loss for trials 48 and 44. For trial 1, the curve for the train- and validation loss are almost identical.

AE - Final Evaluation

As described in Section 5.6, two thresholds were set on the reconstruction loss based on the number of sequences that remains after filtering. Threshold 1 is the 98th percentile of the reconstruction loss for the ATP-ICE test sequences. Threshold 2 was computed by maximizing the difference between the proportion of remaining true- and false-positive sequences. Table 6.4 and Table 6.5 show the main results of the final evaluation using threshold 1 and threshold 2 respectively. Ideally, the proportion of remaining ATP-ICE sequences should be high and the proportion of remaining AFP sequences should be low. The performance of each trial was similar in general. The best performance was achieved with hyperparameters from trial 44. Note, the AFP class contains sequences from both the AFP-DRY and AFP-SB classes.

Table 6.4: Proportion of remaining sequences for each class after filtering using threshold 1.

	% of remaining sequences			
Trial	ATP-ICE (Test)	AFP-DRY	AFP-SB	AFP
40	98%	90%	91%	90%
14	98%	72%	82%	73%
48	98%	73%	82%	74%
7	98%	81%	87%	82%
44	**98%**	**71%**	**80%**	**72%**
1	98%	72%	80%	73%

Table 6.5: Proportion of remaining sequences for each class after filtering using threshold 2.

	% of remaining sequences			
Trial	ATP-ICE (Test)	AFP-DRY	AFP-SB	AFP
40	87%	73%	79%	73%
14	87%	57%	65%	57%
48	90%	58%	70%	58%
7	98%	81%	87%	82%
44	**90%**	**56%**	**68%**	**57%**
1	90%	56%	68%	57%

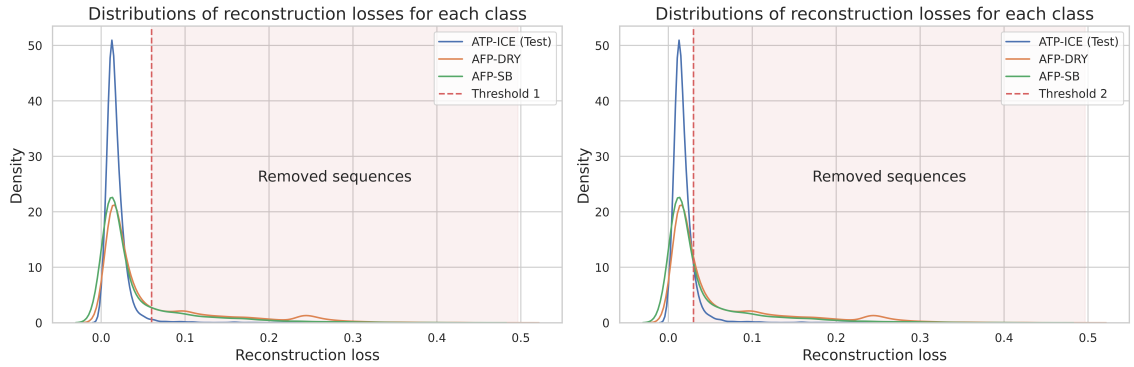

Figure 6.10: Distributions of the reconstruction losses for each class and two different thresholds.

For trial 44, the distributions of the reconstruction losses for each class and the two thresholds (red dotted lines) described in Section 5.6, are shown in Figure 6.10. In this figure, the x-axis corresponds to the reconstruction loss, whereas the y-axis is the density as in the figures previously shown. Sequences receiving a reconstruction loss above the threshold are assumed to be false positives.

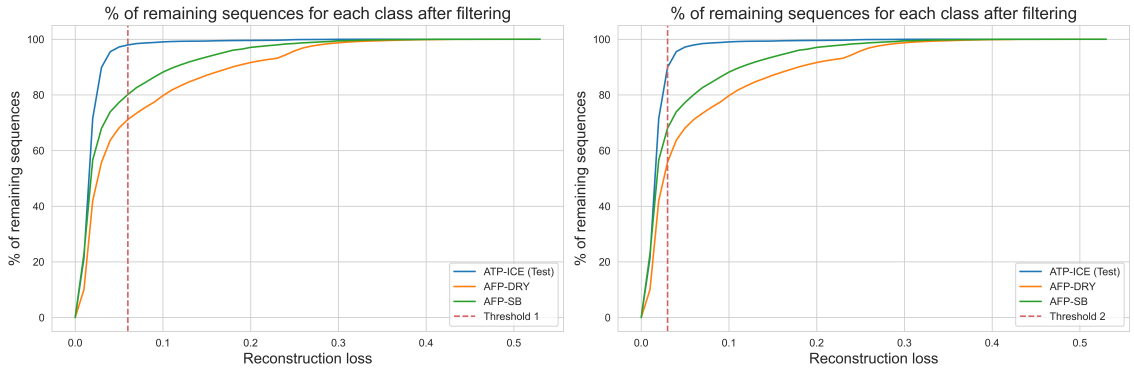

Figure 6.11: Proportion of remaining sequences after filtering using both thresholds.

The proportion of remaining sequences for each class after filtering using thresholds 1 and 2 is shown in Figure 6.11. Since the distributions of the reconstruction losses are similar for both true- and false-positive sequences, the percentage of remaining false-positive sequences will inherently be relatively high while still retaining many true positives.

6.2 Active Learning

This section covers the three experiments conducted with the active learning approach. The two first experiments focused on testing different features and parameters for the classifier models. The third experiment was dedicated to applying active learning and testing which size and parameters performed best. Finally, the best-performing classifiers from this approach were selected and then used for the final evaluation as false positive filters. For each experiment, both LSTM and GRU layers were tested for handling the sequential inputs. Additionally, static sequences were removed in half of these tests In Table 6.7, the dynamic column indicates the experiments in which only dynamic sequences were included.

AL - Experiment 1
Table 6.6 shows the results of the baseline experiment for the LSTM classifier. For this experiment, the following three features were used: friction coefficient (FC), friction quality (FC), and sequence length (SL).

Table 6.6: Results from active learning experiment 1.

	Parameters			Metrics	
	Model	Features	Dynamic	F1 Score	Accuracy
1	LSTM	FC and FQ	No	0.45549	0.51074
2	**GRU**	**FC and FQ**	**No**	**0.57960**	**0.61501**
3	LSTM	FC and FQ	Yes	0.45831	0.51014
4	GRU	FC and FQ	Yes	0.58514	0.58581
5	LSTM	FC,FQ and SL	No	0.54312	0.53541
6	GRU	FC,FQ and SL	No	0.55992	0.54449
7	LSTM	FC,FQ and SL	Yes	0.54779	0.53133
8	GRU	FC,FQ and SL	Yes	0.53306	0.58066

From these results, it is clear that the baseline using only sequential features is not enough, with test 2 reaching an accuracy of 61%. When using the GRU model, the performance was better without adding sequence length or removing static sequences. For the LSTM models, however, this gave a clear improvement in the results. Overall, GRU seems to have performed slightly better than LSTM when using only the sequential features.

AL - Experiment 2
The next experiment was dedicated to using a varying number of features from the data set. The main features tested here were the vehicle features, weather features, and finally training models using all features. The results of this experiment are shown in Table 6.7.

Unlike the first experiment, the LSTM models performed better in this experiment. Test 6 resulted in the highest F1 score and accuracy. However, the difference between test 6 and the second best test 9, which used a GRU network, is negligible. Similar to the first experiment, the best-performing model was trained on both static and dynamic sequences. The models using only vehicle features (tests 1-4) performed significantly better than the baseline models from experiment 1. Additionally, the

Table 6.7: Results from active learning experiment 2.

	Parameters			Metrics	
	Model	Features	Dynamic	F1 Score	Accuracy
1	LSTM	Vehicle Feature	No	0.78212	0.77482
2	GRU	Vehicle Feature	No	0.77317	0.77255
3	LSTM	Vehicle Feature	Yes	0.77337	0.77119
4	GRU	Vehicle Feature	Yes	0.76687	0.77194
5	FNN	Weather Features	No	0.86303	0.84943
6	**LSTM**	**All Features**	**No**	**0.89082**	**0.88423**
7	GRU	All Features	No	0.88551	0.88136
8	LSTM	All Features	Yes	0.88731	0.88226
9	GRU	All Features	Yes	0.89019	0.88287

model using only weather features (test 5) had a performance almost comparable to the classifiers with full features (test 6-9).

AL - Experiment 3
The best-performing classifier from experiment 2 was used for hyperparameter tuning using Optuna before initializing the active learning iterations. The best hyperparameters are presented in Table 6.8.

Table 6.8: Active learning - Best hyperparameters.

Hyperparameter	**Values**
Optimizer	Adam
Learning Rate	7.45e-05
Batch_size	128
Hidden size	256
Emd dropout	0.6
FC dropout	0.0

Classifiers for each test in experiment 3 were trained using these hyperparameters and then used in active learning iterations. The results from these tests are shown in Table 6.9. In general, similar to the initial models or slightly worse or better. However, it quickly became clear that the performance did not increase as new data samples were added.

Figure 6.12 shows an example of an active learning iteration using uncertainty sampling. The F1-score and accuracy never increased beyond the results of the initial model without data samples added. In some of the tests, the performance after active learning is even slightly worse, such as in test 7. Even though the active learning iterations do not increase the performance, the starting size of the data set does. Changing the sampling method results in similar performance as seen in Figure 6.13 where margin sampling is used instead. In these figures, the y-axis represents the metrics whereas the x-axis is the data samples used when training the models.

Table 6.9: Results from active learning experiment 3.

	Parameters		Initial		Final	
	Start Size	Sampling Method	F1 Score	Accuracy	F1 Score	Accuracy
1	300	Uncertainty Sampling	0.761	0.756	0.761	0.764
2	300	Margin Sampling	0.773	0.767	0.766	0.766
3	600	Uncertainty Sampling	0.799	0.790	0.796	0.792
4	600	Margin Sampling	0.793	0.782	0.798	0.794
5	1500	Uncertainty Sampling	0.823	0.817	0.804	0.804
6	1500	Margin Sampling	0.822	0.817	0.809	0.808
7	3000	Uncertainty Sampling	0.846	0.839	0.830	0.828
8	3000	Margin Sampling	0.841	0.836	0.843	0.840

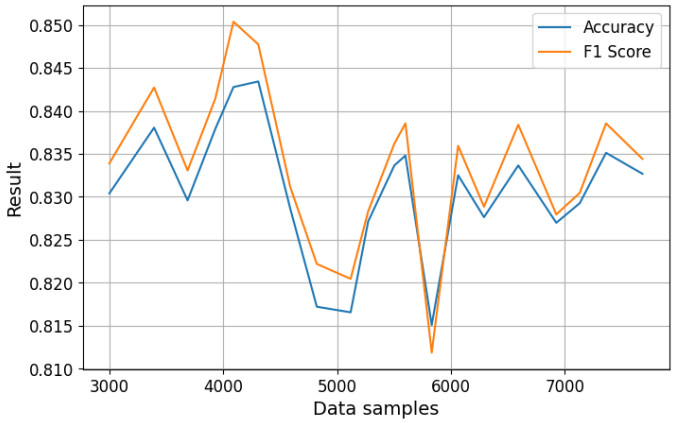

Figure 6.12: Active learning iteration for least certain sampling, test 7.

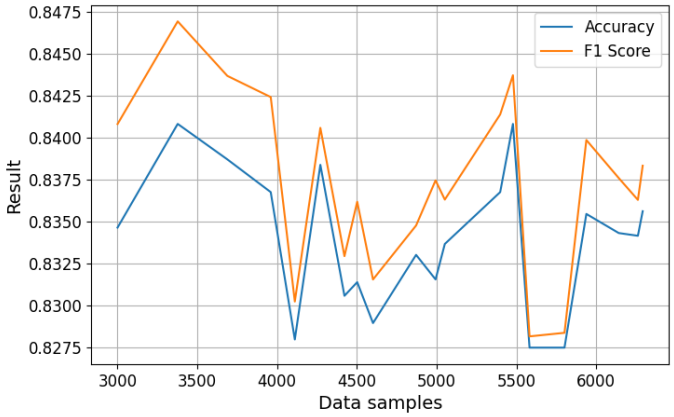

Figure 6.13: Active learning iteration for margin sampling, test 8.

As a result of the active learning not improving the performance, the best-performing classifiers from experiment 2 were used instead for the final evaluation. These models were the best-performing in tests 6 and 9, in addition to test 5 the model only used weather features to test the difference.

AL - Final Evaluation

The final step of the active learning approach was to evaluate the performance of the models. Table 6.10 shows the remaining proportion of sequences for each class after filtering. From these results, it is noticeable that the model only using weather features is able to remove the largest proportion of false positives while retaining the most true positives sequences. However, this result is misleading and will be discussed in the next chapter.

Table 6.10: Proportion of remaining sequences for each class after filtering using the classifier models.

	% of remaining sequences			
Trial	ATP-ICE	AFP-DRY	AFP-SB	AFP
Full features GRU	88.82%	26.63%	4.38%	25.35%
Full features LSTM	81.31%	27.07%	4.96%	25.80%
Weather features	**91.21%**	**3.75%**	**3.37%**	**3.72 %**
Vehicle features LSTM	80.10%	35.47%	8.62%	33.92%

7

Discussion and Conclusion

This chapter provides a discussion of the results and important parts of the project including the available data and how the models were evaluated. Section 7.1 and 7.2 discuss the results of the autoencoder- and active learning approaches respectively. Section 7.3 discusses challenges with the available data. In Section 7.4, the evaluation of the models is discussed. Section 7.5 discusses the ethical and legal considerations for this project. Section 7.6 provides suggestions for future work. Finally, Section 7.7 concludes the thesis.

7.1 Autoencoder

The results from the first experiment of the autoencoders show that the mean MSE losses are generally small for each class, as shown in Table 6.1 and Table 6.2. The reason for the low losses is that MSE squares the difference between the input and reconstructed values. Squaring a difference that is less than one will make it smaller. However, there were noticeable differences in the mean reconstruction losses between true- and false-positive sequences, which is especially noticeable for setting 1 and 4 in Figure 6.1.

For settings 1 to 4 when the friction quality was excluded and the autoencoder was trained solely on the friction coefficients, the mean MSE loss was larger for the assumed false positive sequences (AFP-DRY and AFP-SB). Combining both the friction coefficient and friction quality as features (settings 5 to 8) made it more difficult for the models to distinguish between the assumed true- and false positive sequences on average. Additionally, for settings 5 to 8, the mean MSE loss was mostly lower for the AFP sequences which is the opposite of what is intended. Unlike the friction coefficient, the friction quality is an ordinal variable ranging from 4 to 7. However, the friction quality was treated as a continuous variable similar to the friction coefficient. Additionally, during the analysis of the sequential vehicle data, the friction quality did not seem to change much within the sequences. Hence, this might be the reason why the models struggle to distinguish between sequences when including friction quality as an additional feature.

Furthermore, the results from the first experiment indicate that including both short and constant sequences (settings 1 and 5) reduces the mean MSE loss for both true- and false-positive sequences. The reason for this is probably because constant and short sequences are easier to reconstruct than longer and dynamic sequences.

Additionally, short and constant sequences might be similar in each class. Hence, including these sequences will make the mean MSE loss smaller for all classes and thus making it difficult to distinguish between true- and false positive sequences.

As can be seen in both Table 6.1 and Table 6.2, a small mean MSE loss for ATP-ICE sequences did not necessarily result in a large difference between mean MSE loss of the ATP-ICE and AFP sequences. Setting 4 resulted in the largest mean MSE loss difference but not the smallest mean MSE loss for the ATP-ICE sequences. The smallest mean MSE loss for the ATP-ICE sequences was achieved in setting 1. As described in the theory chapter, autoencoders might learn the identity function resulting in low reconstruction losses for any input. This typically occurs when the embedding size is equal to or larger than the size of the input. The embedding size was probably too large in the first experiment which might be an explanation for the low reconstruction losses. The reason why setting 4 resulted in the largest mean MSE loss difference might be because longer sequences that are dynamic vary more and are less similar between the ATP-ICE and AFP sequences.

In most cases, for both the GRU- and LSTM autoencoders, the mean MSE loss for the assumed false positive sequences during dry weather conditions (AFP-DRY) is larger compared to sequences on speed bumps (AFP-SB). The reason for this could be that the causes for the assumed false positive sequences in the AFP-DRY class are more in number resulting in more varying types of sequences with different properties. Sequences in the AFP-SB class are probably more consistent in their structure since they are all generated close to speed bumps. Additionally, speed bump sequences might be more similar to sequences generated on ice. In that case, since the autoencoders are trained on ATP-ICE sequences, the reconstruction loss will inherently be lower for AFP-SB sequences. Furthermore, both the ATP-ICE and AFP-DRY classes can still contain speed bump sequences even after the data preprocessing. The reason is that the speed bump data did not contain all speed bumps in Gothenburg and some speed bumps are temporally placed due to construction. If the training data contained speed bump sequences, the autoencoder will learn how to reconstruct those sequences with low reconstruction loss. This could therefore also be a reason for the lower reconstruction loss for the AFP-SB sequences.

In general, the LSTM autoencoder performed slightly better in terms of larger mean MSE loss differences compared to the results of the GRU autoencoder, specifically for setting 1 and 4. Note that the mean MSE loss for the ATP-ICE classes is similar for both autoencoders even though the mean MSE loss differences are increased using LSTM. One reason for this could be that the LSTM autoencoder found and learned other patterns in the sequences since it can capture long-term dependencies better than GRU autoencoders. These patterns might deviate more from the AFP sequences compared to the patterns found with the GRU autoencoders.

Scaling the MSE loss in the second experiment did not change the distributions of the reconstruction losses nor the training as shown in Figure 6.4 and Figure 6.5. This result was expected since scaling the loss will only result in scaled gradients.

The third experiment for the autoencoder approach showed that it is difficult to

simultaneously achieve a low validation loss and a high loss difference between the ATP-ICE and AFP-DRY sequences with the hyperparameters selected for tuning. A high validation loss generally corresponds to a high loss difference. This means that when the model fits the training data well, it will on average perform worse in distinguishing true- and false-positive sequences. The reason for this is probably because the autoencoder had learned a close approximation of the identity function in those cases. Hence, any input to the model will generate a low reconstruction loss. As shown in Table 6.3, trials 40 and 7 used an embedding size of 128 which resulted in low validation loss and loss difference which can be seen in Figure 6.8.

Hyperparameter tuning using Optuna was an extremely time-consuming process. Trials could not be pruned since pruning using multiple objectives has not yet been implemented in Optuna. Instead, each model within each trial was trained for each epoch. Additionally, training GRU- and LSTM-based autoencoders are in general time-consuming especially when the training data is large, which was the case in this project. Instead of using Optuna, a less time-consuming process would be to manually test combinations of hyperparameters. However, the search space would be reduced and the most optimal set of hyperparameters would most likely not be found.

Due to the high time complexity of the hyperparameter tuning process, the number of trials was set to 50 which can be considered to be small. This parameter should be large enough such that Optuna can find the most optimal hyperparameters. A too-small number of trials might have resulted in a less optimal set of hyperparameters.

The final evaluation of the autoencoder with the best-performing hyperparameters showed that it is possible to filter out some of the assumed false positive sequences while retaining a high amount of assumed true positive sequences. The proportion of remaining sequences for each class after filtering was similar using hyperparameters for each selected trial. As shown in Table 6.4 and Table 6.5, the best result was achieved using the hyperparameters from trial 44. The reason for this might be because the embedding size was low enough such that the decoder of the autoencoder learned how to reconstruct the ATP-ICE sequences better. As previously discussed, trials 40 and 7 resulted in a low loss difference between ATP-ICE and AFP-DRY sequences probably because of the large embedding size.

It is important to note that the assumed true- and false-positive sequences were grouped based on assumptions. It is impossible to determine how accurate these assumptions were without ground truth. Additionally, the data used to create the classes, including the weather data and road data, might have contained errors and missing data. Accurate measurement of precipitation levels is challenging due to the natural variability and some of the locations of speed bumps were not present in the road data.

In Figure 6.10, it is clear that most of the assumed false positive sequences received low reconstruction losses. There are two main explanations for this. First, the structure might be similar for a majority of the assumed true- and false-positive sequences. Second, the training data might have contained false positive sequences but should ideally only contain true positive sequences. The autoencoder would

then have learned to reconstruct both true- and false-positive sequences and thus resulting in low reconstruction loss for both classes.

The thresholds for the final LSTM autoencoder were set based on the 98th percentile of the ATP-ICE reconstruction loss and the threshold that yields the largest difference between the amount of remaining true- and false-positive sequences. Using the second threshold resulted in more assumed false positive sequences being removed. Conversely, the number of assumed true positive sequences remaining was higher using the first threshold. One can argue that retaining a high amount of assumed true positive sequences is important since they most likely correspond to situations that resulted in a loss of traction. If it is certain that all sequences in this class are true positives, then the proportion of remaining sequences in the ATP-ICE class after filtering should be 100%. However, since the sequences were grouped based on assumptions, some of the sequences were probably placed in the wrong class. A more realistic filtering would therefore allow some sequences in the ATP-ICE class to be removed, i.e. using the second threshold which resulted in 43% removed AFP sequences and 90% remaining ATP sequences. It is however difficult to determine if the removed sequences in the ATP-ICE class corresponded to true- or false-positive situations. This needs to be investigated further since removing actual true positives should be avoided.

As previously discussed, the trained autoencoders in this project can find patterns in the assumed true positive sequences. However, it is difficult to interpret what these patterns are. For example, when slipping on ice, the estimated friction coefficient might drastically drop to a low value and continue to be low through the entire sequence until the friction increase. Conversely, when driving on a speed bump, the friction coefficient might fluctuate more between high and low friction coefficients. The autoencoder might have picked up on these patterns but it is difficult to determine if that is the case. It is also possible that the autoencoder found other patterns in the sequences that are difficult to interpret.

7.2 Active Learning

Before discussing the results of the active learning part it is important to note that the data set used in this part was pseudo-labeled using the weather and road data. It is therefore difficult to make real conclusions from these results even though they might seem promising.

The results from experiment 1 of the active learning approach were somewhat expected. As previously mentioned, the performance of these classifiers was poor. Interestingly, the best-performing model was a GRU model using only the friction coefficient and friction quality as features. However, when the sequence length was added as an additional feature, the performance dropped. Conversely, for the LSTM classifiers, dropping static sequences and adding the length increased the accuracy. A hypothesis is that having more distinguishable sequences and knowing their length helped the LSTM classifiers differentiate between the classes better. It could be that the GRU picks up some patterns missed by the LSTM models and including

the length might have introduced bias which results in slightly worse performance for the GRU classifiers. This would imply that the increased results in the later tests for the LSTM could be a result of bias and not an improvement.

In Experiment 2 it is clear that adding weather-related features significantly increases the performance since both the vehicle features and weather features contain temperature and yield over 20% increase in accuracy. Additionally, the difference between the best-performing model from test 6 and the model with only weather features is only around 5% accuracy. From these results, it is clear that this labeled data set has a large bias toward weather features. However, this is only natural since the pseudo-labeling was done using a combination of weather and road data based on our assumptions. One could argue that the task of detecting slippery roads itself has an inherent bias towards weather features and therefore this might not be as problematic as it seems. However, when the labeled data contains bias it is only natural that a model with the same bias will perform well. It is therefore not possible to conclude how well these classifiers would work on the real data where without this bias. The uncertain nature of the pseudo-labeled data set means that these results can be misleading. Therefore, labeled data and testing would be required before any conclusions could be achieved.

In the last experiment, the use of active learning to increase performance was tested. After applying active learning iterations, there was no increase in performance. However, changing the initial size of the data set yielded increased performance meaning that the issue is likely related to not selecting relevant data samples. There are four possible explanations for the poor performance: the lack of ground truth data, the weather bias in the models, the lack of human annotation, and finally the sampling methods used. Human annotation was excluded since the human expertise required for the was to be available. Additionally, trying to label the data using our knowledge would then be similar to how we created the labeled data set and would therefore be redundant and time-consuming. Other data sampling methods such as estimated model change might work better for the classifiers used in this project compared to the uncertainty sampling methods used. However, these would instead be more computationally heavy and harder to implement. Additionally, the selected samples might not be the issue but the annotation. A solution would then be to use human expertise or create a system to more accurately label the data instead of relying on the classifier. The weather bias of the models is more challenging to fix. However, changing the weights for weather features might help reduce the impact bias of these features.

7.3 Available Data

The available data for this project was limited in some aspects which had an impact on the results and evaluation of the proposed solution. Some of them are listed below:

- **Unlabeled data:** The lack of labeled data in this project introduced two main challenges. First, since the data was unlabeled, only unsupervised learning meth-

ods could be considered. Additionally, active learning which is a semi-supervised learning technique, was also used as an alternative approach. Supervised learning techniques have been studied a lot and have shown promising results in classification tasks and could have been used if the data had been labeled. Second, evaluating the performance of a trained model in unsupervised settings is challenging. The main reason for this is that there is nothing to compare the output from the model with. Even in unsupervised learning, it is common to have at least a small set of labeled data to validate against. However, with no labeled data, the evaluation is instead based on assumptions that might not reflect reality well. Therefore, further testing is required to assess the performance of the proposed solution in real scenarios.

- **Few informative features:** The available data lacked informative features that are needed to determine if data points correspond to true- or false-positive situations. The estimated friction coefficient and weather features are not enough. Even if there are perfect weather conditions for ice formation on the road surface and the estimated friction coefficient is low, it does not necessarily mean that the vehicle slipped on ice. There are many other potential causes for the estimated friction coefficient to be low e.g. driving on a speed bump or making a sharp turn.
- **Preprocessed data:** The vehicle data had already been filtered based on the friction quality which means that not all friction values from the vehicles were available. This results in incomplete sequences that might not contain enough information to infer if the sequence corresponds to a slippery or non-slippery situation.
- **Temporal issues:** In this project, sequences were generated since the vehicle data did not contain enough informative features to determine if a situation corresponded to a true- or false-positive situation. This approach assumes that the sequences are in the correct order. However, the only available time variable was the time in which data points were created on the database. The creation time might not have been an accurate time variable for the creation of the sequences. This could have resulted in data points being placed in the wrong order within the sequences.
- **Spatial issues:** The sequences were not only created based on the creation time but also the locations of each data point. During the analysis of the vehicle data, it was discovered that some data points had the same location but different creation times. This could have resulted in sequences containing data points from different situations.

7.4 Evaluation

Since the available data was completely unlabeled in this project, the trained models could not be evaluated against ground truths. Instead, the models were evaluated based on assumptions of what a reasonable performance of the models is considering

the goal and how the data sets were constructed. Hence, it is uncertain how the models would perform when deployed in real-world settings.

The performance of autoencoders for anomaly detection is commonly evaluated using metrics relying on labeled data such as accuracy, F1 score, or precision and recall. Since ground truth data was not available in this project, these metrics could not be used. Instead, the autoencoders were evaluated based on the mean reconstruction loss and the difference between the mean reconstruction loss for the assumed true- and false-positive sequences. The mean reconstruction loss and the difference might not be good indicators of the autoencoder performance because the mean is affected by outliers. If some sequences receive a high reconstruction loss, the mean reconstruction loss will be increased. For this reason, the median was also computed but did not change the outcome and was therefore not included in the final results. Instead of relying on the mean or median of the reconstruction losses, it is also possible to compute the full width at half maximum (FWHM) of the reconstruction loss distribution as in [52]. Due to the limited time of the project, this was not investigated further and is left for future work.

Similarly, evaluating the classifiers used in the active learning approach was challenging as well. The data sets were pseudo-labeled based on the data analysis using both weather- and road data. The classifiers were evaluated using common metrics such as accuracy and F1-score. However, these metrics can be misleading in this setting since it indicates the performance on the pseudo-labeled data and not the ground truth. Before the proposed solution is implemented in an SRA system and used in production, it is therefore important that extensive testing is done to ensure safety.

The final evaluation of the models from the autoencoder- and active learning approach was done by computing the number of remaining sequences after filtering for each class. This indicates how well the models can filter out the assumed false positive sequences while retaining a high amount of assumed true positive sequences. However, even if the models can remove many assumed false positive sequences, it does not necessarily mean that the number of false positive SRAs will be reduced. It could be that the SRA system already would neglect the removed data points, for example in the aggregation process. Therefore, a more robust evaluation of the models would be to use them alongside an SRA system and compare the number of alerts generated with and without the filter. This was the ultimate goal of this project, but it could not be accomplished due to limited time.

7.5 Ethical and Legal Considerations

There are several ethical concerns regarding this project that are important to take into account in the development of a solution to the problem of false positive SRAs. First, for safety applications such as the SRA system, it is important that the solution is reliable and works as intended. For example, the amount of true positive alerts should still be high after filtering such that the overall performance is not decreased. If the SRA system fails to send warnings in real slippery situations,

the main purpose of the system is not fulfilled. Conversely, an SRA system that generates many false positive warnings will be less trustworthy which can result in drivers neglecting the received warnings, including true positives. Therefore, the proposed solution must be thoroughly tested before being deployed.

Second, privacy is an important aspect that needs to be taken into account, including what type of data is being collected. The data sets used in this project were therefore anonymized during the data preprocessing by removing the vehicle identification number (VIN). Hence, this makes it impossible to determine from which vehicle the data points were sent.

Finally, the proposed solution should be interpretable such that the developer can understand the causes and reasoning of decisions made by the model. A highly interpretable model is more trustworthy since it is known how it behaves and what leads to its predictions. Additionally, interpretability is especially important in the process of improving the model performance and detecting bias. In this project, it is clear that including weather conditions as features induce bias in the classifiers. However, bias can sometimes be difficult to detect and can result in side effects that were not expected. Deep learning models such as those developed in this project tend to be less interpretable due to their complexity. There are however methods such as SHapley Additive exPlanations (SHAP) [53] that can be used to increase the interpretability of these types of models.

7.6 Future Work

There are three main paths for future work on implementing a false positive filter in an SRA system. The first option is to continue with similar data used in this project and try to improve the proposed solution. The second option is to gather labeled data such that supervised- or semi-supervised learning techniques can be used. Finally, the third option is to train machine learning models on data from other sources.

1. Improving the proposed solution using similar data
The most challenging approach would be to use data similar to what was used in this project. From our experience, it is clear that there are limitations with this data and that labeled and more informative features would be needed to create a robust solution. If more time were available for this project, the data preprocessing could have been improved. First, the data was clustered based on both temporal and spatial properties of the vehicle data to form sequences. However, it could be that these sequences contained data points from multiple situations. This needs to be investigated further to ensure that each sequence corresponds to one single situation. It is possible that more accurate sequences would result in better performance of the machine learning models. Second, the sequence grouping process could be improved in the preprocessing step. The data sets were created based on assumptions in which weather conditions true- and false-positive sequences are likely to occur and if the sequences were close to speed bumps or gravel roads. These assumptions could be more accurate with expert knowledge. This would result in more accurate ATP and

AFP classes. Hence, the training data would contain more true positive sequences and fewer false positive sequences.

Another possible path would be to explore different model architectures and more complex models. For the autoencoder approach, variational autoencoders (VAEs) have shown promising results in anomaly detection [54],[55],[56] and could potentially perform better than the GRU- and LSTM autoencoders developed in this project. Additionally, autoencoders based on transformers [8] could also be investigated further. Transformers are known to be more efficient in both training and prediction which would be beneficial when using the model in production. Additionally, to reduce the training and execution time of the autoencoders, it is possible to pack the padded sequences so the model does not need to unroll for the padded time steps.

For the active learning approach, there is not much potential with the current data. Experimenting with additional sampling methods, for example by using the expected model change maximization (EMCM) [57] might prove to work better. However, the core issue here is the lack of labeled data. It is therefore uncertain if improving either the classifiers or the active learning sampling would lead to much better results. Exploring active learning with additional resources might therefore be a more worthwhile endeavor.

2. Gathering labeled data
Labeled data is often both time-consuming and expensive to collect. However, the labeled data set do not need to be large, even a small labeled data set would be useful for both of the approaches used in this project. Since the data sets were constructed based on assumptions, it is uncertain if sequences were assigned to the correct class. For the autoencoder, a small set of labeled data could be used to make sure that the training data consists of a majority of true positive sequences. The autoencoder would then be able to learn how to reconstruct only true positive sequences such that false positive sequences receive a higher reconstruction loss, thus making them easier to separate.

Active learning is a type of semi-supervised learning technique where it is essential to have at least a small initial labeled data set. In this project, we experimented with the possibility of relying on pseudo-labeling instead of labeled data. However, the results showed that implementing this is not worth the effort. The reason is that the initial data set in active learning sets the basis for how the classifier will perform. It is probable that using the same methods but starting with small high quality labeled data set can achieve better results. This is not guaranteed since there is also a possibility that other changes to this approach would be needed for a practical false-positive filter.

Labeled data would also make the evaluation of the models easier and more robust. It would then be possible to assess the actual performance of the models. However, additional testing to make sure the models work as intended would still be needed.

3. Exploring other data sources
The final option for future work is to train machine learning models on data from

other sources and use additional features such as vehicle speed and acceleration. More informative features of the situation would lead to more accurate models. As mentioned previously in the discussion, the vehicle data had previously been processed. It would be interesting to use less processed vehicle data since it could potentially reveal more differences between true- and false-positive sequences.

In addition to the vehicle data used in this project, it is also possible to use data from other sensors such as lidar and radar to determine the road condition as in [58],[59],[60]. Another approach could be to train a convolutional neural network (CNN) on images of the road surface to detect black ice, similar to what was done in [61]. Finally, it would be interesting to combine multiple data sources and machine learning models in a complete system. Ensemble models might work well for reducing false positive SRAs since many possible causes of false positive alerts might require different types of models and data to be classified accurately.

7.7 Final Conclusion

The main goal of this thesis was to reduce the number of false positive alerts generated by an SRA system using two machine learning techniques: autoencoder and active learning. Additionally, a sub-goal was to find potential root causes of these false positive alerts. During the analysis of the available data, the hypothesis that weather conditions correlate with the number of generated alerts was confirmed. Another interesting discovery was that speed bumps seem to cause false positive alerts.

The best-performing autoencoder was able to remove 43% of the assumed false positive sequences while retaining 90% of the assumed true positive sequences. The amount of removed assumed false positives might be less than desired. However, with the available data and resources, it at least shows that filtering assumed false positive sequences is possible. Due to the limited time for this project, the removed sequences could not be analyzed further. However, this would be necessary to better understand the root causes of alerts and if the autoencoder work as intended.

Since the data sets were created based on assumptions of when true- and false-positive sequences are likely to occur, the data sets might not only contain sequences corresponding to their respective class. Additionally, some of the assumed false positive sequences were most likely similar in structure to the assumed true positive sequences. These issues could be the reason why the reconstruction loss for the majority of the assumed false positive sequences was low.

The results from the classifiers might look promising. However, since the data was both grouped and labeled using weather data, the models are highly biased and it is therefore uncertain how their performance would be compared to the ground truth. The final autoencoders on the other hand only used friction coefficients meaning that the data itself does not contain any direct bias towards weather conditions. The results from the autoencoders can therefore be seen as more reliable.

As previously mentioned in the discussion, there are several options to improve

the proposed false positive filter developed in this project. The main issue that complicated both the development and evaluation of the proposed solution had to do with the available data. Most important would therefore be to gather labeled data and more informative features of the situations. Even a small set of labeled data would make the problem less challenging. Then, the performance of the active learning approach would most likely be improved and the evaluation of both the autoencoders and classifiers would be more reliable.

Bibliography

[1] Trafa. "Road traffic injuries." (2023), [Online]. Available: https://www.trafa.se/en/road-traffic/road-traffic-injuries/. Accessed: 2023-05-30.

[2] J. Börgeson and A. Stålheim, "Sensor data fusion for road friction estimation," M.S. thesis, Chalmers University of Technology, 2010.

[3] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection," *ACM Computing Surveys*, volume 54, number 2, pages 1–38, Mar. 2021. DOI: 10.1145/3439950. [Online]. Available: https://doi.org/10.1145%2F3439950.

[4] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pages 305–316. DOI: 10.1109/SP.2010.25.

[5] M. F. Lima, B. B. Zarpelão, L. D. H. Sampaio, J. J. P. C. Rodrigues, T. Abrão, and M. L. Proença, "Anomaly detection using baseline and k-means clustering," in *SoftCOM 2010, 18th International Conference on Software, Telecommunications and Computer Networks*, 2010, pages 305–309.

[6] N. Vallez, A. Velasco-Mata, and O. Deniz, "Deep autoencoder for false positive reduction in handgun detection," *Neural Computing and Applications*, volume 33, number 11, pages 5885–5895, 2021.

[7] O. I. Provotar, Y. M. Linder, and M. M. Veres, "Unsupervised anomaly detection in time series using lstm-based autoencoders," in *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, IEEE, 2019, pages 513–517.

[8] S. Tuli, G. Casale, and N. R. Jennings, *Tranad: Deep transformer networks for anomaly detection in multivariate time series data*, 2022. arXiv: 2201.07284 [cs.LG].

[9] P. Rajput, M. Chaturvedi, and V. Patel, "Road condition monitoring using unsupervised learning based bus trajectory processing," *Multimodal Transportation*, volume 1, number 4, page 100 041, 2022.

[10] J. M. Celaya-Padilla *et al.*, "Speed bump detection using accelerometric features: A genetic algorithm approach," *Sensors*, volume 18, number 2, page 443, 2018.

[11] G. W. Flintsch, K. K. McGhee, E. D. L. Izeppi, and S. Najafi, "The little book of tire pavement friction," *Pavement Surface Properties Consortium*, volume 1, pages 1–22, 2012.

[12] R. Rajamani, N. Piyabongkarn, J. Lew, K. Yi, and G. Phanomchoeng, "Tire-road friction-coefficient estimation," *IEEE Control Systems Magazine*, volume 30, number 4, pages 54–69, 2010.

[13] J. Hall, K. L. Smith, L. Titus-Glover, J. C. Wambold, T. J. Yager, and Z. Rado, "Guide for pavement friction," *Final Report for NCHRP Project*, volume 1, page 43, 2009.

[14] S. d, S. Jaichandar, G. Rajan, and S. Govindan, "Coefficient of friction in different road conditions by various control methods - an overall review," *International Journal of Mechanical and Production Engineering Research and Development*, volume 8, pages 9–20, Jan. 2018. DOI: 10.24247/ijmperddec20182.

[15] D. Harwood, R. Blackburn, B. Kulakowski, D. Kibler, *et al.*, "Wet weather exposure measures," Turner-Fairbank Highway Research Center, Tech. Rep., 1988.

[16] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[17] X. Wang, Y. Zhao, and F. Pourpanah, "Recent advances in deep learning," *International Journal of Machine Learning and Cybernetics*, volume 11, pages 747–750, 2020.

[18] T. O. Ayodele, "Types of machine learning algorithms," *New Advances in Machine Learning*, volume 3, pages 19–48, 2010.

[19] D. Greene, P. Cunningham, and R. Mayer, "Unsupervised learning and clustering," *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, pages 51–90, 2008.

[20] M. Fuchs and W. Höpken, "Clustering: Hierarchical, k-means, dbscan," in *Applied Data Science in Tourism: Interdisciplinary Approaches, Methodologies, and Applications*, Springer, 2022, pages 129–149.

[21] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *ArXiv Preprint ArXiv:2004.00433*, 2020.

[22] B. Settles, "Synthesis lectures on artificial intelligence and machine learning," in *Active Learning*, Springer Cham, 2012, pages 1–11.

[23] R. M. Monarch, *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Simon and Schuster, 2021.

[24] A. Freytag, E. Rodner, and J. Denzler, "Selecting influential examples: Active learning with expected model output changes," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pages 562–577, ISBN: 978-3-319-10593-2.

[25] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, volume 415, pages 295–316, 2020.

[26] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pages 2623–2631.

[27] Optuna. "Multi-objective Optimization with Optuna." (2018), [Online]. Available: `https://optuna.readthedocs.io/en/stable/tutorial/20_recipes/002_multi_objective.html` (visited on 06/17/2023).

[28] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, volume 96, 1996, pages 226–231.

[29] A. Krogh, "What are artificial neural networks?" *Nature Biotechnology*, volume 26, number 2, pages 195–197, 2008.

[30] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, volume 6, number 12, pages 310–316, 2017.

[31] A. F. Agarap, "Deep learning using rectified linear units (relu)," *CoRR*, volume abs/1803.08375, 2018. arXiv: `1803.08375`. [Online]. Available: `http://arxiv.org/abs/1803.08375`.

[32] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, volume 8, pages 4806–4813, 2019.

[33] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?" *Neural Computation*, volume 16, number 5, pages 1063–1076, 2004.

[34] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate Research*, volume 30, number 1, pages 79–82, 2005.

[35] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, volume 26, number 1, pages 98–117, 2009.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv Preprint ArXiv:1609.04747*, 2016.

[37] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, Elsevier, 1992, pages 65–93.

[38] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv Preprint ArXiv:1308.0850*, 2013.

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, volume 27, 2014.

[40] I. Goodfellow, Y. Bengio, and A. Courville, "Sequence modeling: Recurrent and recursive nets," *Deep learning*, pages 367–415, 2016.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, volume 9, number 8, pages 1735–1780, 1997.

[42] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *ArXiv Preprint ArXiv:1601.06733*, 2016.

[43] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, IEEE, 2017, pages 1597–1600.

[44] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, 2014, pages 4–11.

[45] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, series Proceedings of Machine Learning Research, JMLR Workshop and Conference Proceedings, volume 27, Bellevue, Washington, USA: PMLR, 2012, pages 37–49.

[46] B. X. Yong and A. Brintrup, "Do autoencoders need a bottleneck for anomaly detection?" *IEEE Access*, volume 10, pages 78 455–78 471, 2022.

[47] E. J. Hastings, J. Mesit, and R. K. Guha, "Optimization of large-scale, real-time simulations by spatial hashing," in *Proc. 2005 Summer Computer Simulation Conference*, Citeseer, volume 37, 2005, pages 9–17.

[48] Trafikverket, *Lastkajen*, version 6.0. [Online]. Available: `https://lastkajen.trafikverket.se/login`.

[49] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig, "Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru," *Journal of Artificial Intelligence and Soft Computing Research*, volume 9, number 4, pages 235–245, 2019.

[50] R. Cahuantzi, X. Chen, and S. Güttel, "A comparison of lstm and gru networks for learning symbolic sequences," *ArXiv Preprint ArXiv:2107.02248*, 2021.

[51] R. C. Aygun and A. G. Yavuz, "Network anomaly detection with stochastically improved autoencoder based models," in *2017 IEEE 4th international conference on cyber security and cloud computing (CSCloud)*, IEEE, 2017, pages 193–198.

[52] E. Ordway-West, P. Parveen, and A. Henslee, "Autoencoder evaluation and hyper-parameter tuning in an unsupervised setting," in *2018 IEEE international congress on big data (BigData congress)*, IEEE, 2018, pages 205–209.

[53] S. Mangalathu, S.-H. Hwang, and J.-S. Jeon, "Failure mode and effects analysis of rc members based on machine-learning-based shapley additive explanations (shap) approach," *Engineering Structures*, volume 219, page 110 927, 2020.

[54] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special lecture on IE*, volume 2, number 1, pages 1–18, 2015.

[55] H. Xu *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, 2018, pages 187–196.

[56] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, volume 3, number 3, pages 1544–1551, 2018.

[57] W. Cai, Y. Zhang, and J. Zhou, "Maximizing expected model change for active learning in regression," in *2013 IEEE 13th international conference on data mining*, IEEE, 2013, pages 51–60.

[58] G. Sebastian, T. Vattem, L. Lukic, C. Bürgy, and T. Schumann, "Rangeweathernet for lidar-only weather and road condition classification," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2021, pages 777–784.

[59] A. Bystrov, E. Hoare, T.-Y. Tran, N. Clarke, M. Gashinova, and M. Cherniakov, "Sensors for automotive remote road surface classification," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, IEEE, 2018, pages 1–6.

[60] S. M. Sabery, A. Bystrov, P. Gardner, A. Stroescu, and M. Gashinova, "Road surface classification based on radar imaging using convolutional neural network," *IEEE Sensors Journal*, volume 21, number 17, pages 18 725–18 732, 2021.

[61] H. Lee, K. Hwang, M. Kang, and J. Song, "Black ice detection using cnn for the prevention of accidents in automated vehicle," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2020, pages 1189–1192.

A

Appendix

A.1 Autoencoder Experiment 3 - All trials

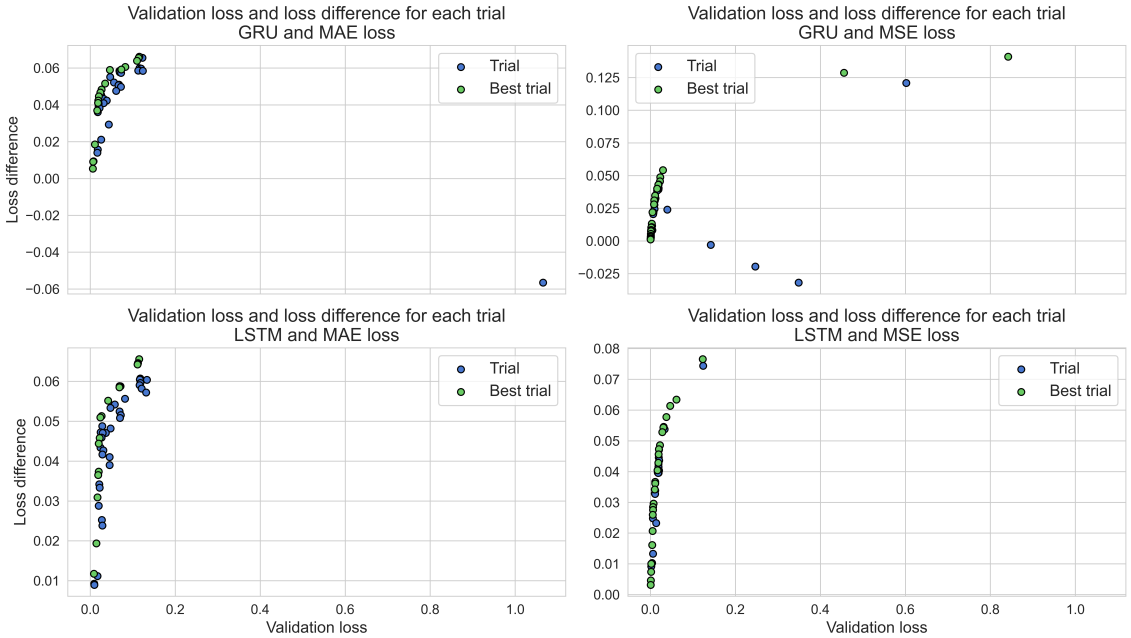

Figure A.1: The final validation loss and loss difference between ATP-ICE val. and AFP-DRY for both the GRU- and LSTM autoencoder using MAE and MSE as loss function.

A.2 Autoencoder Experiment 3 - Hyperparameter importance

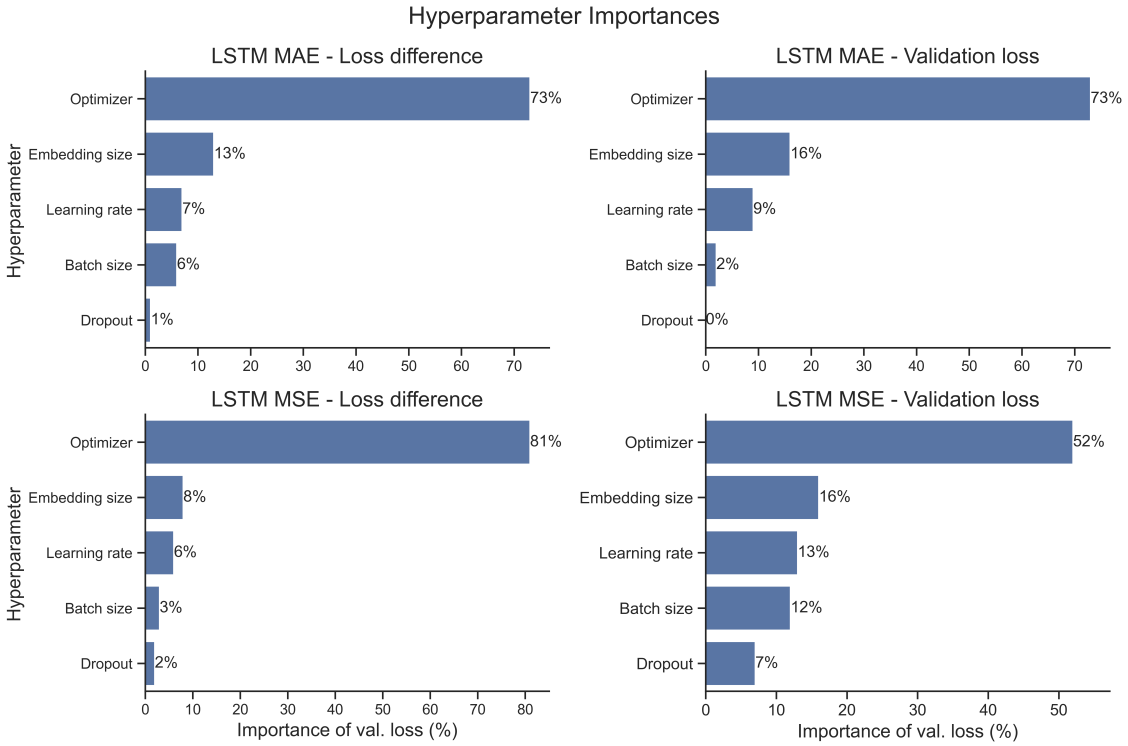

Figure A.2: Hyperparameter importance for each model tuned using Optuna.