CHALMERS
UNIVERSITY OF TECHNOLOGY

Driver Behavior Profiling on Smartphone Data using Machine Learning Methods

Master's thesis in Computer Science: Algorithms, Languages and Logic.

Aryan Iranzamini and Jakob Lindström

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018
EX017/2018

Master's thesis EX017/2018

Driver Behavior Profiling on Smartphone Data using Machine Learning Methods

ARYAN IRANZAMINI
JAKOB LINDSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Driver Behavior Profiling on Smartphone Data using Machine Learning Methods
ARYAN IRANZAMINI
JAKOB LINDSTRÖM

Supervisor: Irene Yu-Hua Gu, Department of Electrical Engineering
Advisor: Oliver Brunnegård, Autoliv
Examiner: Irene Yu-Hua Gu, Department of Electrical Engineering

Master's Thesis EX017/2018
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Driver Behavior Profiling on Smartphone Data using Machine Learning Methods
ARYAN IRANZAMINI
JAKOB LINDSTRÖM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Driver behavior is a significant factor when it comes to traffic safety. There is therefore a need to understand what typical behavioral patterns exist and how to derive them. It is also important to understand how contextual factors affect driver behavior. The thesis has therefore been centered around investigating both how machine learning-based methods can be used to estimate driver behavior pattern, and how contextual variables affect these behavior patterns.

We propose two machine learning methods for estimating driver behavior patterns. One approach is based on deep learning, and the other one is based on driver norms. The results show that both methods yield similar results for data without taking contextual factors into account. Further research is needed, to be able to conclude the effects of contextual factors on driver behavior profiles.

Keywords: Driver behavior, Deep learning, Machine learning, Clustering, Latent subspace, Unsupervised learning, Thesis.

Acknowledgements

We would like to start off by thanking our company supervisor Oliver Brunnegård for providing us with endless guidance and support throughout the thesis project. We are grateful for the amount of freedom we got to tackle the thesis problem, while still getting feedback and being pushed towards the right direction.

We would also like to show our gratitude to our academic supervisor Irene Yu-Hua Gu at the Electrical Engineering department at Chalmers University of Technology. Thank you for your feedback and your valuable inputs throughout the project.

Lastly, many thanks to all the wonderful colleges at Autoliv that have helped us and given us insight into many areas of the thesis work.

Aryan Iranzamini & Jakob Lindström, Gothenburg, May 2018

Contents

List of Figures

List of Tables

1
Introduction

In over 90 percent of road crashes the critical reason of the crash is attributed to the drivers [2]. As a consequence, it would be very beneficial to be able to identify drivers who engage in unsafe driving habits, as well as providing just the right driver coaching to change risky behavior.

With today's information and communications technology, or ICT for short, devices such as smartphones can be used to collect data about the user's spatio-temporal information. This data can in return be used to map other contextual information regarding the user. In a vehicle setting, this would mean that we can derive both the vehicle's dynamics, as well as the contextual setting the vehicle is operating in.

To understand driver behavior, one has to understand the contextual information the driver is operating under. This is because different driving conditions requires a different set of actions from the driver. Therefore to be able to understand different driver behavior profiles, one has to incorporate contextual information, such as speed limits, weather conditions and so on, into the analysis. This area of research has for a long time been of significance, and with the rapid development of today's technology, this opens up new ways for analyzing driver behavior.

1.1 Background and related work

Dong et al. published a paper concerning characterizing driver style using deep learning [3] in 2016. The paper considers the task of identifying different drivers using models based on convolutional neural networks and recurrent neural networks. The data used was based on collected GPS data. The results were compared with gradient boosting decision tree on handcrafted features which the proposed methods outperform when the number of classes increase. The report concludes that deep learning is a good tool for learning driver styles from GPS data.

A way of detecting risky driving events through smartphone sensors and external data was proposed by Castignani et al. [4]. They proposed a system that infers events through fuzzy logic based on the sensor data from smartphones. With the combination of the inferred events and external data, such as weather information and maximum speed limit for the corresponding road, they score each driver to determine their corresponding risk factor. This risk factor consists of three profiles, namely normal, moderate and aggressive driver behavior. This paper shows one us-

age that relates to driver behavior by combining inferred events and external data.

Research conducted by Warren. J et al., [5] showed how driving behavior could be derived from data coming from a smartphone. From this data they created a statistical model over driver behavior norms from 500 different drivers for different road segments. From this data they could conclude norms regarding speed and maneuverability in certain areas of San Francisco. By clustering on the driver data, they grouped together drivers with similar driving patterns. From the driving norms they could flag behaviors in the clusters that deviated from the norm and conclude different driving behaviors. The obtained driver behavior profiles from their clustering consisted of drivers that were potentially aggressive, normal or slower than average. This research paper shows one way of deriving driver behavior based on driving norms and data solely from smartphones.

Zhang Li and Chen Cai proposes an unsupervised method of finding driver behavior patterns using GPS data from various sources [6]. From the GPS data they derived certain features such as speed, location and time which allowed them to compare driving speeds between drivers at various locations and different time periods of the day. This research paper shows an example of how unsupervised methods without any prior knowledge about the drivers can find patterns that distinguishes different types of driver behaviors. They also mention that future work for similar and more extensive projects could be combining the type of data they had with more data from other heterogeneous sources. This would serve as a combination of components that describe behavioral patterns for drivers.

An investigation of different machine learning algorithms and different types of smartphone sensors for driver behavior profiling was conducted by Júnior, Jair Ferreira, et al [7]. The goal of the investigation was to determine which machine learning algorithms and types of sensors performed best for classification tasks. The data they used throughout the investigation was sensor data based on accelerometer, linear acceleration, gyroscope and magnetometer. They conclude that the best sensors for detecting different driving events, such as an aggressive brake or a non-aggressive event, were the gyroscope and the accelerometer, and the machine learning algorithm that performed best was random forest, with neural networks being a close second.

1.2 Problem addressed

Since driver behavior is one of the most important aspects of traffic safety, this raises the question how one can derive different driver behavior profiles. It is also of significance to know what characterizes these different behavior profiles. Furthermore, since the driving context affects each driver's behavior, it is important to understand how driver behavior and driving context together affect driver behavior profiles. This leads to the main research question.

Main research question: How can one derive driver behavior profiles? What characterizes these profiles? How does additional contextual data affect the profiles?

1.3 Aim

DBPs are represented by characteristic traits and behavioral patterns in traffic of a group of drivers. The purpose of this project is to develop a driver profiling algorithm that uses the information from a driver monitoring system to define distinct DBPs. We will also through literature studies obtain a deeper understanding of what adjustments are required to improve the performance of the algorithm.

1.4 Delimitations

The scope of the thesis work will only consist of taking into account behavioral patterns and traits which can be derived from the driver monitoring system data. In other words, an individuals driving behavior is limited to acceleration, deceleration and turning of the vehicle.

2

Overview of background and methods

The following section will give an overview of different methods, algorithms and other important concepts used throughout the thesis work. The theory behind the different algorithms will be presented along with general suggestions on how and when to use them.

2.1 Supervised Learning

Supervised learning is a field of machine learning where we have a dataset with some sort of labeled data or data with a ground truth. The goal of the supervised algorithm is to learn a mapping from a set of inputs x to a set of outputs y for its given task. We also want this mapping from x to y to be as general as possible so that when the algorithm sees new data, it can still perform well. In other words, we have some input x and some output y, and the goal of the supervised algorithm is to learn a mapping from x to y such that it can predict the right output for new unseen data.

The notion of it being *supervised* comes from the fact that when the algorithm is trying to learn a mapping from x to y, it knows the true values for each data sample. It can therefore adapt itself accordingly to the labeled data. In supervised learning there are different types of tasks. These tasks are *Regression tasks* and *Classification tasks*.

Regression Tasks
Regression tasks are tasks where we try to learn a mapping $x \mapsto y$, where x is a set of inputs and y is a continuous variable.

Classification Tasks
Classification tasks are tasks where we try to learn a mapping $x \mapsto y$, where x is a set of inputs and y is a class label. For classification tasks we need to at least have two classes. Tasks with only two class labels are often referred to as binary classification problems. Tasks with more than two classes are referred to as multi-class problems.

2.2 Unsupervised Learning

Unsupervised learning is a machine learning task where we use data without labels or ground truth and instead try to find an underlying structure in the data [8]. Since unsupervised learning differs from supervised learning due to the fact that no labels are present, the approaches of solving problems are very different between these two methods. In supervised learning we are trying to learn a function approximation from a set of inputs to a set of outputs, while in unsupervised learning we are trying to find some latent structure in the data. Also, considering that unsupervised learning does not have any ground truth, one has to resort to heuristic measures when asserting validity.

There are many different unsupervised learning tasks, some examples of the more common ones are *Clustering tasks* and *Anomaly detection tasks*.

Clustering tasks
A clustering task, is where we have a set of objects and want to group together similar objects. The notion of similarity between objects is different for different clustering algorithms, there are therefore many different clustering algorithms that suit different types of data.

Anomaly detection tasks
Anomaly detection is the task of trying to identify outliers or a set of objects/events that do not conform with a usual observed pattern. One can utilize unsupervised learning methods here if the dataset consists of mostly normal behavior.

2.3 Artificial neural networks

Artificial neural networks are mathematical models that are inspired by biological neural networks to some degree [9]. They consist of artificial neurons that input and output values and can through connections to other neurons create a network topology for different computational tasks. They are in other words computational systems that are used for various tasks. There are many different types of neural networks but this project has mainly focused on feed-forward and recurrent neural networks.

2.3.1 Activation functions

Activation functions are used in computational systems, such as neural networks, to map a set of inputs to some output. There are many types of activation functions but the most common ones used are the ones that can express some form of non-linearity. This is because by using non-linear activation functions, a computational system can express more complex functions. The most common activation functions used in neural networks are *ReLu*, *Leaky ReLu Sigmoid*, *Softmax*, *Tanh*.

ReLu
The rectified linear unit (ReLu) is an activation function that maps all negative values to 0 while leaving all positive values unchanged. It is in other words a ramp function and can be expressed as

$$f(x) = x^+ = max(0, x) \tag{2.1}$$

It has been shown that the ReLu activation function has allowed better training of deeper networks in comparison to some of the other traditional activation functions such as sigmoid and tanh [10]. It is also now of 2018, one of the most successful [11] and widely used [12] activation functions.

Leaky ReLu
Leaky ReLu is one popular variation of the ReLu activation function. The difference between the regular ReLu and the Leaky ReLu is that the Leaky ReLu allows for a small non-zero gradient for non-active units [13]. So it is the same as the ReLu function for positive values, but for negative values it can be expressed as

$$f(x) = 0.01x \tag{2.2}$$

Sigmoid
The sigmoid activation function is a special case of the logistic function where $L = 1$, $k = 1$ and $x_0 = 0$. It is defined as

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2.3}$$

One nice property of the sigmoid activation function is that it squashes the output to values in the range of 0 to 1. This makes it convenient for binary classification tasks as it will output values between 0 and 1, which can for instance be the probability for a given class. However, it has the drawbacks that it's gradients tends to go towards zero for deeper neural networks.

Softmax
The softmax activation function is similar to the sigmoid activation function in the sense that it is a generalization of the logistic function to multiple classes. Assume we have n classes, then the softmax output for a class j is defined as.

$$f(x_j) = \frac{\exp(x_j)}{\sum_{n=0}^{n-1} \exp(x_n)} \tag{2.4}$$

Here one can see as well that the sigmoid activation function is a special case of the softmax function for $n = 2$.

Tanh
The tanh activation function is defined as

$$f(x) = \frac{sinh(x)}{cosh(x)} = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \tag{2.5}$$

It can be seen as a scaled version of the sigmoid function since it has the same function shape, but it is scaled between the values -1 and 1 instead. If the data has a mean around zero and a standard deviation of 1, this activation function has shown to be more beneficial in terms of training efficiency for neural networks than the sigmoid function [14].

2.3.2 Feed-forward neural networks

Feed-forward neural networks are networks where you have neurons or nodes in a series of layers where each layer is connected to its adjacent layer. The first layer is known as the input layer where the information flows in from, and the last layer is known as the output layer where the network's output flows out from. The layers in between are known as hidden layers.

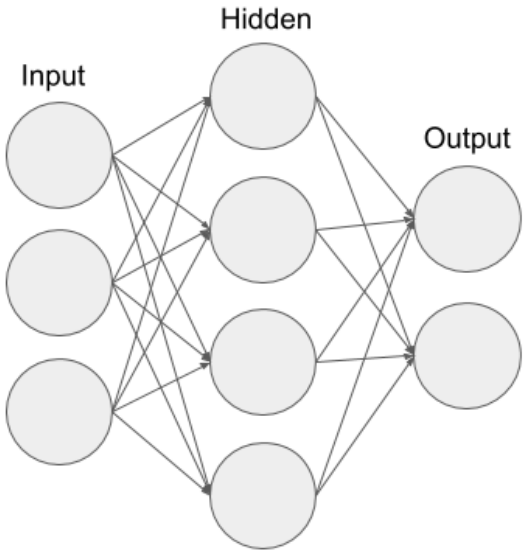


Figure 2.1: An example of a feed-forward neural network.

Looking at Figure 2.1 we can see that each node has multiple inputs and outputs. Each line in the figure is denoted as a connection, meaning that information can flow between two nodes in this direction. The connection between the nodes have weights which is a numerical value associated with them, and each layer of neurons has a bias which is another numerical value associated with the whole layer. The output from each neuron in a layer is an activation function calculated on the input from each node in the previous layer multiplied with their respective weights and bias. When the data in the input layer has propagated through all of the hidden layers we will have received our output.

More formally, let g be an activation function and $z^{[l]}$ its argument, where l denotes the layer. Then the output $a^{[l]}$ of each neuron can be described by Equation 2.6.

$$a^{[l]} = g(z^{[l]}) \tag{2.6}$$

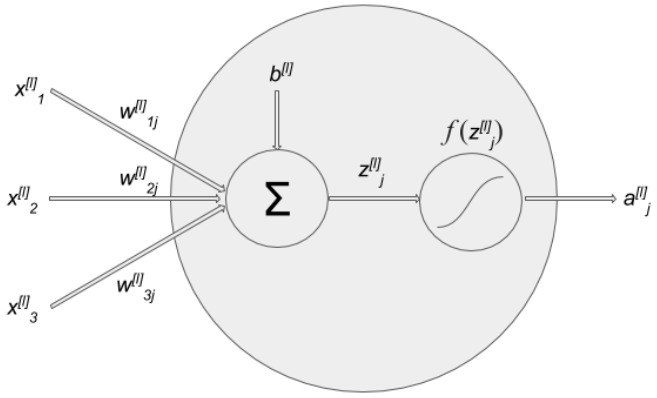


Figure 2.2: An example neuron from a layer l having three inputs and j outputs. f is an activation function, $w_1^{[l]}$, $w_2^{[l]}$ and $w_3^{[l]}$ are the weights and b is the bias.

$$z^{[l]} = w^{[l]T} * a^{[l-1]} + b^{[l]} \tag{2.7}$$

In Equation 2.7 $a^{[l-1]}$ is the input to the neuron fed from the previous layer, $w^{[l]}$ is the weight matrix and $b^{[l]}$ is the bias of the layer. An example of a neuron can be seen in Figure 2.2. It shows a neuron from a layer l with three inputs and j outputs.

2.3.3 Backpropagation

The learning part of a neural network is characterized by minimizing the error of its output. This is done by backpropagation, which essentially is the act of calculating the gradients of the error with respect to the parameters of the neural network. The error of a network is expressed by a cost function, which is a function of the output and the expected output given a particular input. By using the chain rule the gradients $\frac{\partial H}{\partial w^{[l]}}$ and $\frac{\partial H}{\partial b^{[l]}}$, of a given cost function H, for a layer L can be calculated using the gradients from the previous layer $L+1$, as can be seen in Equation 2.8. Also, in the equation the same notation is used as in Equation 2.6 and Equation 2.7.
Given a cost function H, the error $\delta^{[l]}$ for the output layer l can be described by Equation 2.8.

$$\delta^{[l]} = \frac{\partial H}{\partial z^{[l]}} = \frac{\partial H}{\partial z^{[l+1]}} \frac{\partial z^{[l+1]}}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial z^{[l]}} = \delta^{[l+1]} \frac{\partial z^{[l+1]}}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial z^{[l]}} \tag{2.8}$$

$$\frac{\partial H}{\partial w^{[l]}} = \frac{\partial H}{\partial z^{[l]}} \frac{\partial z^{[l]}}{w^{[l]}} = \delta^{[l]} a^{[l-1]} \tag{2.9}$$

$$\frac{\partial H}{\partial b^{[l]}} = \frac{\partial H}{\partial z^{[l]}} \frac{\partial z^{[l]}}{\partial b^{[l]}} = \delta^{[l]} \tag{2.10}$$

Using Equation 2.8 for the errors $\delta^{[l]}$ of layer l, the gradients with respect to $w^{[l]}$ and $b^{[l]}$ can be calculated, as can be seen in Equation 2.9 and Equation 2.10.

As the last layer has no previous layer the equation for calculating the gradient for this layer is a bit different. Equation 2.11 is used for the last layer and depending on the choice of cost function and activation function in the last layer the partial derivatives $\frac{\partial H}{\partial a^{[L]}}$ and $\frac{\partial a^{[L]}}{\partial z^{[L]}}$ vary.

$$\delta^{[L]} = \frac{\partial H}{\partial z^{[L]}} = \frac{\partial H}{a^{[L]}} \frac{a^{[L]}}{z^{[L]}} \tag{2.11}$$

2.3.4 Recurrent neural networks

A recurrent neural network or RNN for short is a type of neural network with loops in the network structure. This enables the network to make predictions based on previous input data as well and not just the current input data. The difference can analogously be viewed as a human reading a text needs to be able to remember what happened on the previous page to be able to make sense of the current page. The loops in the network structure enables the network to simulate a memory and create a context.

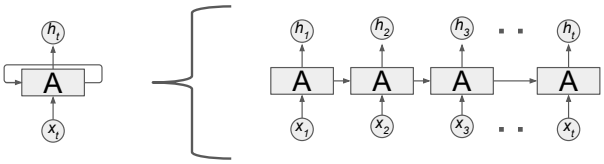


Figure 2.3: A many to many RNN unrolled over the time steps.

An RNN layer consist of multiple cell-units connected in chains feeding cell state information sideways to the adjacent cell-unit and layer output to the next layer. This can be visualized by unrolling the RNN layer and is illustrated in Figure 2.3, where $x_1 \dots x_t$ are the inputs and $h_1 \dots h_t$ the outputs from the different cells at the different time steps for the current layer.

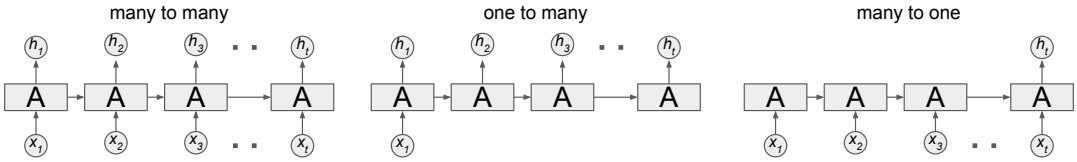


Figure 2.4: Three common network structures of RNNs unrolled.

The RNNs are structured in different ways depending on the task that is being solved. The most common structures are one to many, many to one and many to many. Many to many is for instance used in language translation tasks, the input

can be a sentence in French and the output can be in Swedish. The one to many structure is often used in e.g. image captioning tasks where an image is fed into the RNN and the output is a sentence describing the image. Many to one is for example used in sequence classification tasks, an example task can be to predicting the weather based on the weather from previous days.

There exist different versions of the cell-units with different structure. The most common ones are Long short term memory cells or LSTM, Gated recurrent unit or GRU and the original RNN cell. The LSTM was constructed to address the problems with unstable gradients [15]. The GRU cells is similar to the LSTM and implements mechanisms to deal with the gradients problems but has fewer parameters.

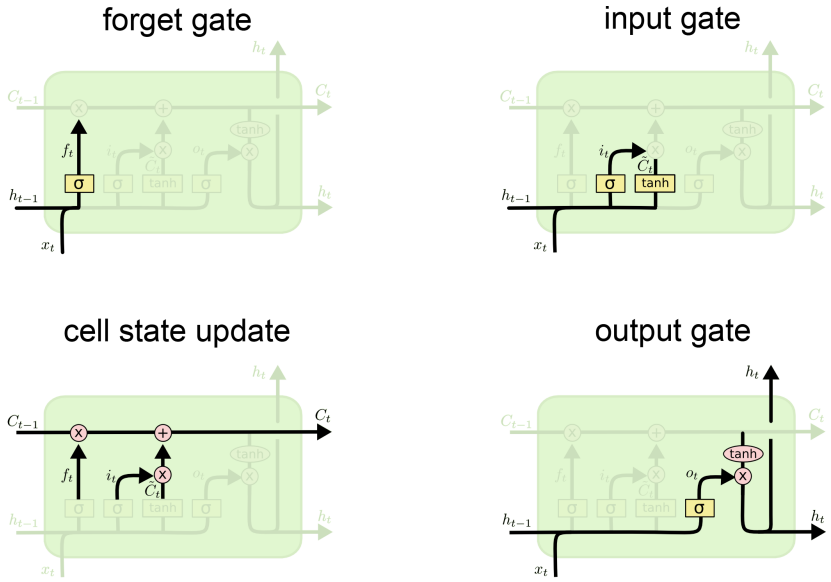


Figure 2.5: The different gates and functions of the LSTM cell. The figure has been made by Christopher Olah [1].

The LSTM cell consists of a cell state and different gate functions which decides how and if the cell state is to be changed. The cell state is the topmost horizontal vector C_{t-1} in the *cell state update* part of Figure 2.5. The other three parts of the illustration, *forget gate*, *input gate* and *output gate*, show the gate functions and are labeled f_t, i_t and o_t. f_t is the forget gate which looks at the input cell state C_{t-1} and decides what to forget and what to keep. This is achieved using a sigmoid neural network with hidden state h_{t-1} of previous cell and input of current cell x_t as input, as can be seen in Equation 2.12.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{2.12}$$

The input gate i_t is described by Equation 2.13 and decides in what places to insert new values from the candidate vector $\tilde{C}_t$. It does so using a sigmoid neural network

just like the forget gate.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.13}$$

The candidate vector $\tilde{C}_t$ is implemented in Equation 2.14 and differ from the other gates by having a tanh instead of sigmoid activation function.

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C) \tag{2.14}$$

Using $\tilde{C}_t$ the update to the cell state is then done with Equation 2.15.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.15}$$

when the new cell state C_t has been calculated the output of the cell h_t is decided by o_t and C_t together in Equation 2.17, o_t is defined in Equation 2.16. In other words it is defined by the sigmoid neural net of previous hidden state h_{t-1}, the input x_t and the tanh of the new cell state from Equation 2.15.

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{2.16}$$

$$h_t = o_t * tanh(C_t) \tag{2.17}$$

2.3.5 Cost functions

Cost functions are mathematical formulas for determining the performance or cost of some output. They can be viewed as an objective function which should be minimized, since minimizing a cost function would in return minimize the total cost. In neural networks for instance, the cost function is a way of measuring the performance of the network. There are many different cost functions but the most common ones will be described below.

2.3.5.1 Mean squared error

The mean squared error MSE is a cost function that measures the difference between an estimate and the actual value. The cost is the error squared and can be described as the following equation

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)$$

where Y is the ground truth and $\hat{Y}$ is the predicted value. This kind of loss function is often found in linear regression tasks. One drawback of this loss function is that it heavily takes outliers into consideration. This is because outliers will have a bigger error and since we square the error, the values will grow much larger than for other points.

2.3.5.2 Cross entropy

The cross entropy cost function H is based on a true probability distribution $p(x)$ and an actual probability distribution $q(x)$ for a variable x.

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

So in other words, it is a cost function that describes the loss between two probability distributions where the true probability distribution is known. This loss function is favored in classification tasks whereas MSE is more commonly used in regression tasks. This is because, as mentioned earlier, the cross-entropy loss is the error between the true probability distribution function and the outputted one and therefore suits probabilistic tasks.

2.3.6 Optimization

The task of training a neural network can be formulated as an optimization problem where the minimum of a particular cost function is to be found.

Gradient descent: One of the more basic optimization algorithms is the gradient descent method or GD which is a building block for the more advanced optimization algorithms used in this thesis. The idea behind GD is to find the minimum of a cost function $H(\theta)$ around a set of parameters θ_t where it is defined and derivable by taking steps in the direction which decrease the value of the function fastest.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta H(\theta_t) \tag{2.18}$$

$$eH(\theta_t) \geq H(\theta_{t+1}) \tag{2.19}$$

By selecting a η small enough Equation 2.19 holds and the new θ_n brings the cost function $H(\theta)$ closer to the minimum. As computing the gradients for the whole dataset can be computationally heavy and also intractable when large datasets are used, which often is the case for deep neural networks, a variant of GD called stochastic gradient descent have been developed.

Stochastic gradient descent: SGD is a variant of the GD optimization, in which the gradients are calculated on one randomly chosen data sample (x^i, y^i) at the time instead of the whole dataset, as can be seen in Equation 2.20. This makes the optimization algorithm feasible for large dataset, but also very stochastic. Because of this SGD can have issues converging and instead overshoot the global minimum. This has lead to the use of a larger subset of the data, also called mini-batch to stabilize the update to the parameters of the network. Equation 2.20 shows SGD on one data sample and Equation 2.21 on a mini-batch of size n.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta H(\theta_t, x^i, y^i) \tag{2.20}$$

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta H(\theta_t, x^{i:i+n}, y^{i:i+n}) \tag{2.21}$$

Momentum: When optimizing highly non-convex cost functions, which training a complex neural network is, it can be hard for SGD to escape local minimum. For instance in a scenario where the gradient landscape has the form of a ravine, as in Figure 2.6, the SGD algorithm would only make small steps in the right direction as it oscillates too much. Momentum [16] adds a resistance to the stochastic parameter update by adding the last update vector through γ as can be seen in Equation 2.22 and 2.23. A common value for the momentum term γ is 0.9, which is suggested by the paper.

$$v_t = v_{t-1}\gamma + \eta \nabla_\theta H(\theta) \tag{2.22}$$

$$\theta_t = \theta_{t-1} - v_t \tag{2.23}$$

Figure 2.6: Momentum has more stable updates and can converge faster than standard stochastic gradient descent.

Adagrad: The Adagrad [17] optimization algorithm adopts the learning rate η to each parameter θ_i individually, this helps learning of features with low occurrence as parameters associated with them get larger updates and parameters associated with more frequent features have smaller updates. In Equation 2.24 the gradients are denoted by g and the learning rate η is changed through the diagonal matrix G_t. The diagonal elements of G_t contains the sum of squared gradients with respect to each parameter θ_i up till time step t. ϵ is a very small number which is added to avoid division by zero.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t \tag{2.24}$$

RMSprop: The RMSprop [18] optimizer was developed to deal with the diminishing gradients problem of Adagrad. As gradients are accumulated during training G_t will grow very large, this in turn leads to a learning rate that eventually reaches zero and the algorithm will stop learning. RMSprop solves this by dividing the learning rate with a decaying average of square gradients, as can be seen in Equation 2.26. Equation 2.26 describes the calculation of the average of squared gradients.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.25}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{2.26}$$

ADAM: The optimization algorithm used in this thesis is the ADAM optimizer [19] which builds upon the GD algorithm and incorporates some other techniques as well. It is considered to be good for training neural networks and is standard for the task in the industry today.

There are three main improvements with ADAM compared to GD. Instead of calculating the true gradient of the cost function an estimate is used. The estimate samples a random subset of the data and thereby decrease the computation time compared to GD where all data is used when the gradient is calculated.

ADAM also makes use of a running average of the gradients and a running average of the square of the gradients as in Equation as in Equation 2.27 and 2.28. This helps the algorithm to converge faster. Because m_t and v_t are initialized to zero they are biased towards zero and hence the bias is corrected by using Equations 2.29 and 2.30.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.27}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.28}$$

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)} \tag{2.29}$$

$$\hat{v}_t = \frac{v_t}{(1 - \beta_2^t)} \tag{2.30}$$

The update to θ_t is then done in Equation 2.31 and by dividing η by the moving average of the square of the gradient a decaying effect is imposed on η and the update to θ_t. Also a small number ϵ is used to prevent division by zero problems.

$$\theta_{t+1} = \theta_t - \frac{\eta}{(\sqrt{\hat{v}_t} + \epsilon)} \hat{m}_t \tag{2.31}$$

2.3.7 Bias-Variance Tradeoff

The bias-variance tradeoff, also known as underfitting and overfitting, is a term used in machine learning to denote two sources of error for a predictive model. The error in a model's prediction can be based on the error from three things, namely the bias, the variance and the irreducible error.

Bias
The bias of a model is the general assumption about the target function. If a model's error is due to high-bias, it is underfitting the target function. The general suggestion to fix a model's high-bias error is to increase the complexity of the model.

Variance
The variance of a model is the change in estimation about the target function when

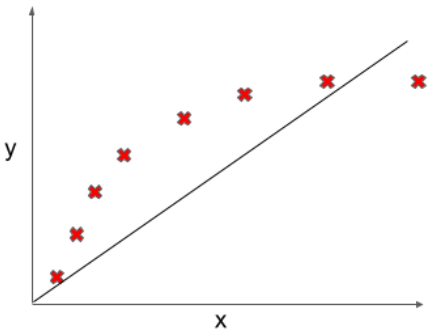


Figure 2.7: A model with high-bias. The model is underfitting the target function.

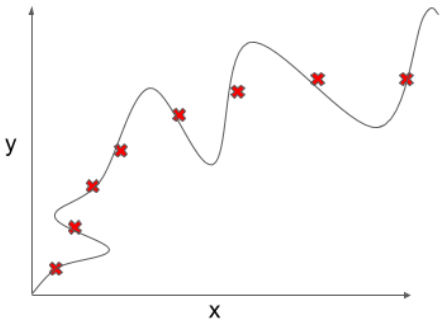


Figure 2.8: A model with high-variance. The model is overfitting the target function.

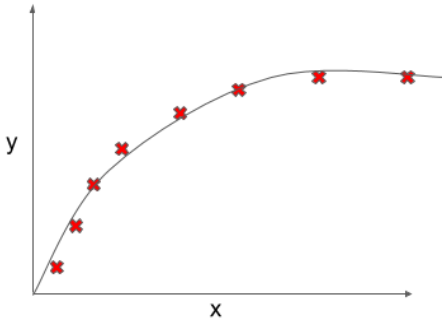


Figure 2.9: A model with low-bias and low-variance. The model has just enough bias and complexity to predict the right target function.

changing training data. A model with a high-variance error is due to it being too complex, which results in overfitting. A too complex model will overfit to random noise which means that it won't learn the real underlying distribution of the data. The general suggestions to fix a model's high-variance error is to use more training data or reduce model complexity.

Irreducible error
The irreducible error is the inherent noise in the problem itself. This error is seen as irreducible since there is almost always a natural variability in the problem that is unpredictable. The opposite of irreducible error is reducible error, which are errors from bias and variance.

2.4 Clustering methods

Clustering or cluster analysis is the task of grouping objects in such a way that objects belonging to the same group are in some measure more similar than objects belonging to another group. The groups are often referred to as clusters and there exists many different algorithms for the task which excel at data with different prop-

erties. As a consequence of how the data is analyzed and not depend on any ground truth data, clustering algorithms are often used in unsupervised data analysis.

2.4.1 K-means clustering

One common clustering technique is the k-means clustering technique which was originally proposed by Lloyd S. [20]. The idea is to group data points into k distinct groups. This is done by first randomly placing out k centers, then assigning data points to the center which they have the least euclidean distance to. Let X=$\{x_0, x_1, \ldots, x_n\}$ be a set of data points with d dimensions and $S = \{s_1, s_2, \ldots, s_k\}$ a set of clusters. The cluster assignment of each data point x can be expressed as choosing

$$argmin_{s_i \in S}\left(\sqrt{\sum_{i=0}^{d} s_i - x_i}\right) \tag{2.32}$$

The centers are then re-positioned to the mean of the data points in their corresponding cluster.

$$s_i = \frac{1}{|s_i|} \sum_{x_i \in s_i} x_i \tag{2.33}$$

The re-positioning of the centers continues until convergence has been reached.

The k-means algorithm is relatively fast compared to other clustering algorithms in the sense that it's complexity time is $\mathcal{O}(\log n)$. However some drawbacks are that one is not guaranteed to get an optimal solution and the algorithm might get stuck in a local minimum. This means that one would get sub-optimal clusters. Also k has to be specified, which means that one either needs to have some idea of how many clusters are optimal or use some heuristic measure to derive an appropriate amount of clusters. It is also sensitive to outliers as they affect the cluster's repositioning each iteration. Since its distance measure is the euclidean distance, it is also sensitive to different scaling variables. Due to its distance measure being the euclidean distance, it is suited well for spherical clusters with similar variance. In summary, it is an easy and relatively fast method to use that is easy to interpret, however it has some drawbacks that are important to keep in mind.

2.4.2 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN), is a density-based clustering algorithm [21]. Density-based clustering means that the algorithm will group together points in certain areas where the data points are more compactly or densely represented.

DBSCAN differs quite a lot from k-means when it comes to clustering different types of distributions, it has the added benefit of being able to cluster non-linear data. Another advantage of this algorithm is that it has a notion of noise and can therefore classify certain points as noise and ignore them during the clustering. In comparison to CLARANS, which is another similar clustering algorithm in the sense that

it is based on spatial data clustering, the authors of the paper show that DBSCAN outperforms it by a factor of over 100 in terms of efficiency.

DBSCAN has many benefits and is very powerful when used correctly and under the right circumstances. To really utilize it correctly, one has to understand how it works. The algorithm takes two parameters as input, namely *minpts* and *eps*. The algorithm then iterates over each point and creates an n-dimensional sphere with the radius *eps*. If a number of *minpts* points are within the sphere, then the center of the sphere becomes a cluster. We then expand the radius of the cluster by doing the same procedure for the points in the cluster, other than the center point. Whenever the procedure does not find *minpts* points, the expansions of the cluster radius stops and the algorithm repeats with a new random point and the whole procedure continues until every point has been visited. Points that are not in a cluster are regarded as noise.

As previously stated, the DBSCAN is very powerful. However, it is really dependent on its parameters. Choosing the radius *eps* requires a good distance function and a good understand of one's data. *Minpts* also requires one to know the data well since this parameter decides what is considered a cluster and what is seen as noise. In summary, DBSCAN is a powerful tool for clustering but requires one to be very familiar with one's data.

2.4.3 Hierarchical clustering

Hierarchical clustering is a clustering method to group together similar data points in an hierarchical manner [22]. There are two strategies for creating these clusters, namely *Agglomerative clustering* and *Divisive clustering*. Both these strategies require a distance function and something called a linkage criteria. The distance function specifies how we measure similarity in forms of distance. This means that a distance function should have a low distance between two similar points and a large distance between two dissimilar points. The linkage criterion specifies how one measures the distance between two groups of points. For instance, does one take the average distance of each point between two clusters or the nearest ones to calculate the distance? The different strategies and the common linkage criteria will be presented down below.

Agglomerative clustering
One strategy for building a hierarchy of clusters is called Agglomerative clustering [23]. It starts by assigning each point as its own cluster, then it merges pairs of clusters by a defined linkage criterion. More thoroughly, every iteration of the algorithm picks the two clusters with the lowest distance between them and merges them. This continues until all clusters are merged. By defining a minimum number of clusters k, one can stop the algorithm when only k clusters are left.

Divisive clustering
Another strategy for building a hierarchy of clusters is Divisive clustering [24]. The

difference between Divisive clustering and Agglomerative clustering is that Agglomerative clustering starts by defining each point as its own cluster, while Divisive clustering starts by assigning all points to the same cluster. Therefore, instead of merging clusters together as Agglomerative clustering does, Divisive clustering splits the clusters based on the dissimilarity between points. By specifying a minimum cluster number k, the algorithm can split the data points to k clusters and then stop.

Single linkage
Single linkage is a linkage criteria where the distance between two clusters is defined by only two points, which is the two closest points between two clusters. This criteria or method is known as the *Nearest neighbor clustering* since only the two closest points are considered. The linkage criterion's distance function $d(i, j)$ between two clusters i and j can mathematically be expressed as

$$d(i, j) = \min_{x_i \in i, x_j \in j} d(x_i, x_j) \tag{2.34}$$

Ward's method
Ward's method is a common linkage criterion used in hierarchical clustering and is an intra-cluster variance minimizing method [25]. Let i and j be two different clusters then the inter-cluster variance $d_{i,j}$ is calculated as following

$$d_{i,j} = d(\{X_i\}, \{X_j\}) = ||X_i - X_j||^2 \tag{2.35}$$

2.5 Dimensionality reduction techniques

High dimensional data can be problematic for some algorithms since there are algorithms with a complexity time that depend on the number of dimensions. It is therefore beneficial to try and reduce the number of dimensions while preserving as much information as possible. It is also much easier to visualize and analyze data with lower dimensions than data with higher ones.

2.5.1 Principal component analysis

Principal component analysis (PCA) uses an orthogonal transformation to reduce the dimension of data [26]. The transformation is defined such that the first component accounts for the highest variance in the data, the second component accounts for the second most variance, etc. More formally it can be expressed as

$$C = \sum_{i=1}^{n} \frac{(x_i - \hat{x})(x_i - \hat{x})^T}{n - 1} \tag{2.36}$$

where C denotes the covariance matrix and $\hat{x}$ the sample mean. Now, we can calculate the eigenvalues λ of C by solving the following equation

$$|C - \lambda I| = 0 \tag{2.37}$$

and we get the corresponding eigenvectors v by solving

$$(C - \lambda I)v = 0 \tag{2.38}$$

Getting the eigenvalues and eigenvectors from the covariance matrix can be thought of as fitting the principal component to the variance of the data. The eigenvalues are coefficients to each eigenvector, by sorting the eigenvectors in order of their eigenvalues, highest to lowest, we sort each principal component in order of variance maintained.

2.5.2 T-distributed stochastic neighbor embedding

T-SNE is a non-linear dimensionality reduction technique [27] which means that it can express more complex polynomial relationships between features than linear techniques.

We start by calculating the conditional probability $p(j|i)$ that the datapoint x_i would pick x_j as its neighbor. x_i would pick x_j as its neighbor in proportion to their probability density under a Gaussian centered at x_i, which can be written as

$$p(j|i) = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-||x_i - x_k||^2/2\sigma_i^2\right)} \tag{2.39}$$

where σ_i is the variance of the Gaussian centered on x_i. By symmetrizing the conditional probability such that $p_{ij} = \frac{p(j|i)+p(i|j)}{2n}$, potential problems with outlying datapoints are prevented.

Let us now denote y_i and y_j as the low-dimensional counterparts of x_i and x_j. We want y_i and y_j to represent the similarities between x_i and x_j. Thus we want q_{ij} to be as close to p_{ij} as possible, where q_{ij} is defined as

$$q_{ij} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq i} \left(1 + ||y_i - y_k||^2\right)^{-1}} \tag{2.40}$$

The difference here is that q_{ij} uses the Student's t-distribution with one degree freedom to model the similarity between the datapoints. To determine how close q_{ij} is to representing p_{ij} we measure the Kullback-Leibler divergence KL between the two probability distributions, which is defined as

$$KL(P_i||Q_i) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.41}$$

A Kullback-Leibler divergence of 0 means that two probability distributions behaves similarly. Thus minimizing the Kullback-Leibler divergence using gradient descent with respect to the points y_i, results in a mapping from x_i to y_i that better retains the similarities between datapoints.

2.6 Evaluation techniques

An evaluation method measures the quality of the clusters through some heuristic measure. A high cluster quality is achieved when the clusters have high intra-cluster similarities and low inter-cluster similarities. In other words, a cluster quality is considered good, for some similarity-measure, if the elements in the same cluster are similar and elements in different clusters are dissimilar.

2.6.1 Silhouette score

Silhouette score is one evaluation method that measures the distance between a point and all other points in the same cluster, and compares it with the distance from the same point to all other points in a neighboring cluster [28]. This implies that a high silhouette score would mean that the points in each cluster are similar to each other and dissimilar to points in other clusters.

If $a(i)$ is the average distance between a point i and all other points in the same cluster, and $b(i)$ is the average distance between the point i and all points in a neighboring cluster, then in mathematical terms the silhouette score is

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{2.42}$$

From the equation we see that $s(i)$ ranges from $[-1, 1]$ where 1 is the desired the score since this would mean that the clusters are considered well according to the silhouette score. By taking the average silhouette score for each point in each cluster, we get the silhouette score for each cluster.

2.6.2 Elbow method

The elbow method is another evaluation method for deciding an appropriate number of clusters for a dataset [29]. Since many clustering algorithms have a hyperparameter k, where k is the number of clusters to use, the elbow method is used to approximate this k.

The method works by measuring the sum of squared errors (SSE) for $k = \{0, 1 \ldots, n\}$ up to some arbitrary number n. The so called 'elbow point' is located where the number of clusters go up but the SSE does not decrease as rapidly as before. In other words the k with the most significant change in the derivative of SSE is chosen. The reasoning behind this is that if $k = m$, where m is the number of points in the data, then we would assign each point to its own cluster. Thus having the minimum SSE possible, but this would be nonsense. Instead, we are looking for a k where adding additional clusters will not decrease the SSE substantially more.
An example of the elbow method and how to find an elbow point is shown in Figure 2.10. It is important however to keep in mind that the elbow point cannot always be unambiguously identified.

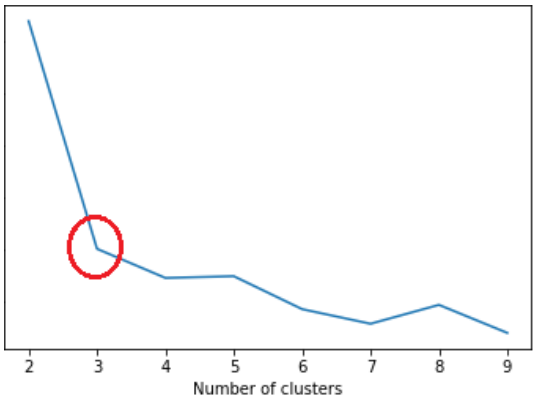


Figure 2.10: An example of the elbow method. The red circle indicates where the elbow point is, which means that this is the optimal number of clusters according to the elbow method.

2.6.3 Empirical analysis

One has to keep in mind that unsupervised clustering methods are exploratory methods. Therefore another way of evaluating cluster results is to use a dimensionality reduction technique and visualize the clusters. But since reducing the number of dimensions can result in some information loss, the visualization often serves as just an approximation of similarities/dissimilarities between data points. A clustering result is considered good if elements in each cluster are clearly a distinct group.

2.7 k-Dimensional Tree

k-dimensional trees or k-d trees as they often are referred to are data structures to store k-dimensional points in. They are useful in search tasks with multidimensional search keys, for instance nearest neighbor search.
The tree is a binary tree where each node contains a k-dimensional point and two subtrees. The subtrees are constructed by dividing the space by the root using a hyperplane at the point of the root node. The hyperplane is chosen so that it is perpendicular to a specific axis of the root point. Points in the space with a larger value than the value of the point in the root node with respect to the selected axis are added to the right subtree. Vice versa, points with lower value are added to the left subtree. The axis is selected at random for the first node and is then altered as each level of the tree is constructed. The point inserted in the node in each of the two subtrees is selected to be the median of the points in each subtree with respect to the selected axis. The time complexity of tree construction is $\mathcal{O}(n \log^2 n)$ if a sorting algorithm with time complexity of $\mathcal{O}(n \log n)$ is used in median selection. Nearest neighbor search can be done in $\mathcal{O}(\log n)$ time.

2.7.1 Nearest neighbor search

The nearest neighbor search in a k-d tree starts by recursively moving down the tree, selecting the closest subtree at each respective axis. When a leaf node has been reach

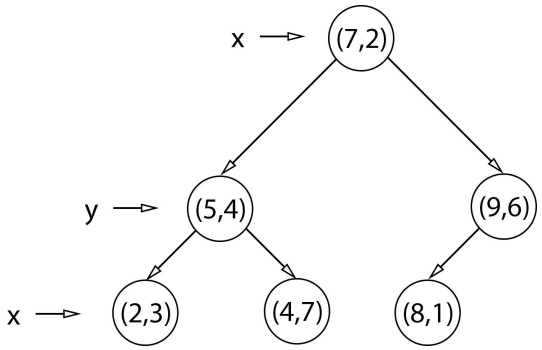


Figure 2.11: A two dimensional k-d tree where the root node's splitting hyperplane is the x-axis at 7. Every point with a value of x less than 7 will be contained in the left subtree and every point with a value of x higher than 7 will belong to the right subtree.

it is marked as a candidate. Now the recursive path is traversed backwards checking if each node is closer than the candidate node. Also to eliminate candidates from the other subtree, a multidimensional sphere is created. The spheres center is at the point being searched for and has a radius of the current lowest distance. If the sphere crosses the hyperplane that splits the root tree then the other subtree also needs to be searched. If this is not the case then the other subtree can be pruned and the algorithm continues up towards the root node.

3
Implementation

The implementation of the driver behavior profiling algorithm can briefly be described using Figure 3.1. The first step is the data pre-processing, which follows this introduction down below. After the pre-preprocessing the two methods for building a cluster space, namely *latent subspace* and *driver norms*, are presented in section 3.2. Lastly the clustering is described in section 3.3.

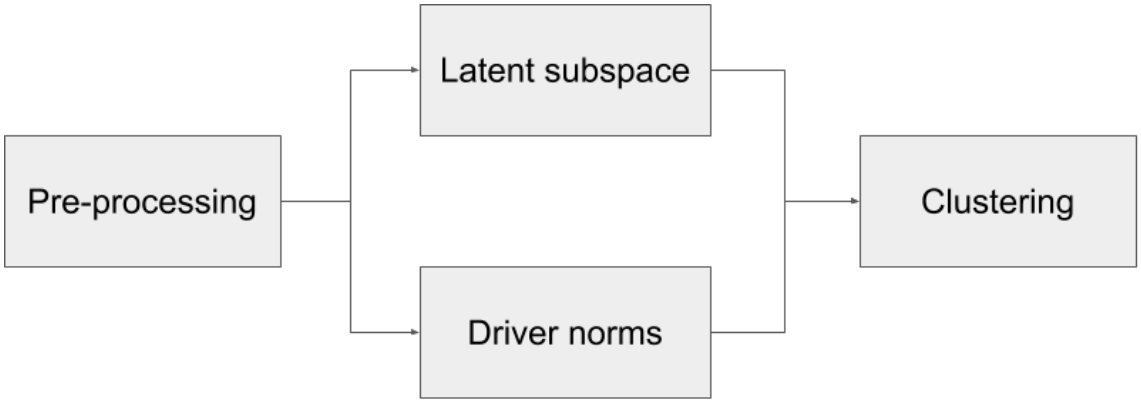


Figure 3.1: Brief overview of the implementation work flow. Data is first preprecessed, then one of the two methods is used to form a space to later cluster on.

3.1 Data set

All data used throughout the thesis work was provided by our industrial partner since the work was done in collaboration with them. The data consists of trips made on a daily basis by 1634 unique drivers from all continents of the world.

Throughout each trip the driver provides accelerometer, gyroscope and GPS information from their smartphone. The sensor data can then be used to derive different driving events. The data was provided in the form of driving events with information on the type of event, start, end, max, min and integral value of the event. Events can e.g. be acceleration, deceleration or a turn to name a few, the complete set of event are listed in Table 3.1. The GPS information allows one to pinpoint the driver's waypoint which gives the possibility to add additional data to provide more information about the driver's situation, such as weather conditions, time of the day, etc.

Table 3.1: The structure and features of the different events are displayed in the table below.

Acceleration
Integral value
Max value
Mean value
Start time
End time

Turn
Angle of turn
Magnitude value
Start time
End time

Boundaries
Start time
End time

Mounted
Start time
End time

Traffic
Start time
End time

Crash
Start time
End time

Idle
Start time
End time

Breakdown
Start time
End time

Anomalies
Start time
End time

In summary, each event has general information which is the same across all event types. This is information regarding GPS-location, start time and end time of the event. Each event also has event specific information. The different events' information is summarized in Table 3.1. The Integral value of an acceleration event is the total change of speed during the event, also the sign of this value decides if the event was a deceleration or an acceleration.

The description for each event is given below:
Acceleration: Whenever a driver accelerates or brakes.
Turn: Whenever a driver turns his or her vehicle.
Idle: If a car stands still for a certain period of time, an idle event is produced.
Breakdown: According to the documentation a breakdown occurs whenever a 'car breaks down', we assume this event is produced when a car stalls on the side of a road.
Boundaries: A boundaries event is produced between orientation changes.
Anomalies: Occurs when phone usage is detected.
Mounted: Tracks whenever the driver mounts the phone.
Crash: This event is produced whenever a driver crashes.
Traffic: Whenever a driver is in traffic, this event is produced.

3.1.1 Additional data based on GPS

By using the GPS information and the time of each event from the user collected data, more contextual data could be added from external sources. The external data sources used are presented in Table 3.2.

The external road data used in training was collected from Open Street Map, which is an open platform for map information [30]. It was inspired by the Wikipedia format

Table 3.2: Additional data based on and GPS.

Open street map data
Highway type
Max speed allowed

Day/Night data
Sunrise time
Dawn time
Dusk time
Sunset time

Table 3.3: The different highway types available is listed in the table, ranging from important larger highways to road types of lesser importance.

Highway type
Motorway
Trunk
Primary
Secondary
Tertiary
Unclassified
Residential
Service

and allows the community to contribute to the available map information. More specifically the highway type and the maximum allowed speed was collected from Open Street Map, as these features were presumed to have an effect on the driving behavior. The speed was converted to km/h and the road type was represented by a categorical value showed in Table 3.3. Also the available daylight was presumed to affect the driver behavior and was represented by a categorical value depending on if it was day or night at the time of the event.

3.1.2 Extracting Open Street Map data

The Open Street Map data was extracted from publicly available API servers using the Overpass QL syntax [31]. The data was provided in the form of nodes, each node has an id and a GPS coordinate. Together a set of adjacent nodes formed a trajectory data type called Way which had different data fields like street name, maximum allowed speed and highway type to name a few. This data was then transformed into an array with the data columns *latitude*, *longitude*, *max speed* and *road type*. This data was then queried using the GPS coordinates of a specific event, the exact GPS coordinate of the events were rarely in the data structure as the possible GPS space is much larger than the actual space mapped in the data. This requires a search algorithm to return the nearest neighbor of a queried GPS coordinate as this is the most likely data for that GPS coordinate. To be able to efficiently find the nearest neighbor in the geographical coordinate system a kd-tree was constructed. This data structure gives a search time complexity of $\mathcal{O}(\log n)$. The training data collected from smartphones was then enriched with the external data using the nearest neighbor search.

3.1.3 Day/Night data

The day/night data in Table 3.2 was obtained by first calculating sunrise and sunset or dawn and dusk from GPS position, date, timezone and solar depression. Using the two times from the calculations, a categorical variable representing whether it was light or dark was created. The variable was then added to the corresponding event.

3.1.4 Data exploration and analysis

A lot of the initial time was spent on data analysis and exploration to get a better understanding of the data. Figure 3.2 shows a plot over the distribution of events for all users. This gave some insight in how the events were distributed. One can see that acceleration events are clearly the event that occurred the most, this is reasonable since an acceleration event can either be an acceleration or a deceleration. The second most common event is turning events. In comparison to these two events, the remaining events occur at a much less frequency.

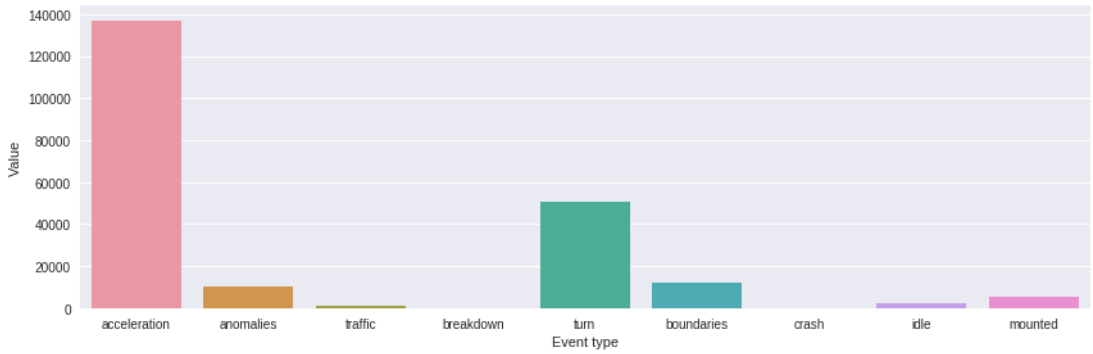


Figure 3.2: The event distributions among all users.

Another important aspect to consider is the distribution of trips among the different drivers. In Figure 3.3 the graph shows that a lot of drivers have a low number of trips recorded. This is important to take into consideration when doing statistical analysis or when trying to apply different machine learning algorithms. It is important for neural networks that each class or label has enough data samples to train on, assuming that the data is representative of each driver, otherwise it will not effectively learn the representation of each class. Also, it is helpful to have data that is as balanced as possible in terms of class distribution, this is because if the data is too skewed then there is a possibility that the neural network will only learn the representation of the majority classes [32].
For models and algorithms that take sequential data into consideration, it is important to take into account the distribution of sequence lengths for various trips. In Figure 3.4 the graph shows that there are a high number of trips with lower amounts of events in them. Also, there are a low number of trips with a very high number of trips in them. This imbalance, the number of events per trip, means that we will have to split up each trip into equally sized trip segments or sequences for it to work with neural networks that utilizes LSTMs.

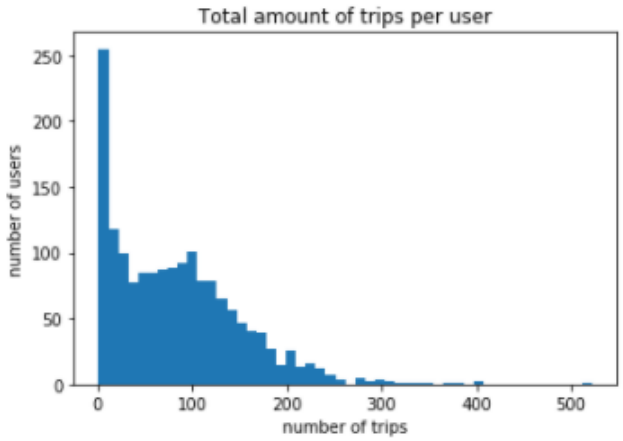


Figure 3.3: The distribution of users over the amount of trips made. Many users have only taken a few trips, as can be seen in the leftmost part of the histogram.

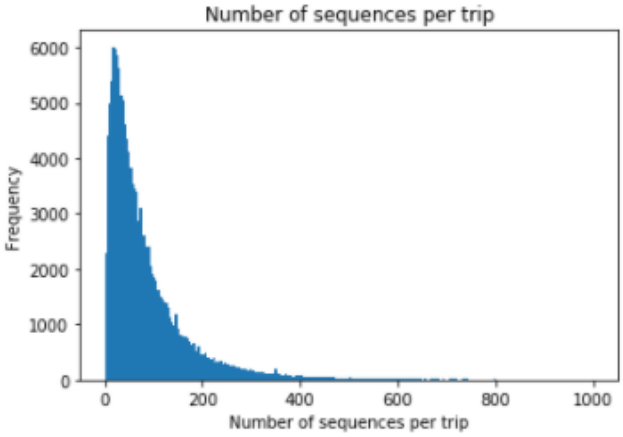


Figure 3.4: Most of the trips have a limited amount of sequences per trip.

3.1.5 Pre-processing

A big part of the project was to pre-process the data to make it compatible with various machine learning algorithms. The pre-processing steps consisted of three major parts, first a reformatting of the data was done so the data had a general structure. Secondly, filtering out users with a low amount of trips since it is hard to draw conclusions about a driver without sufficient data. Lastly, we standardized the data so all features had zero-centered mean with a standard deviation of 1. The workflow for the preprocessing step can be seen in Figure 3.5

3.1.5.1 Reformatting the data

The reformatting of the data started out with sorting each trip's sequence of events by their starting time. The events were sorted based on their time of occurrence since we wanted to preserve the sequential information in each event.

The data had event specific features making it incompatible with different machine

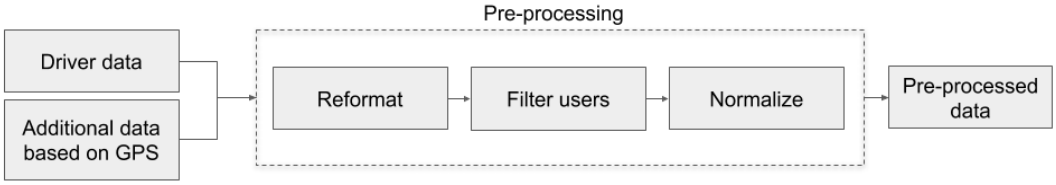


Figure 3.5: The workflow for the pre-processing step. The pre-processing consists of reformatting, filtering and standardizing the input data.

learning algorithms. Formatting the structure of each event into a more general structure made it possible to create similar feature vectors for all types of events.

Listing 3.1: The structure of an acceleration event.

```
{'category': 'acceleration',
    'duration': 12.469,
    'end': '2017-12-13T13:27:34.807+01:00',
    'features': {'magnitude': 0.934},
    'magnitude': 0.934,
    'mean': 1.501,
    'peaks': [{'duration': 12.469,
      'end': '2017-12-13T13:27:34.807+01:00',
      'integral': -69.115,
      'is_positive': False,
      'max': 4.671,
      'mean': 1.501,
      'start': '2017-12-13T13:27:22.338+01:00'}],
    'start': '2017-12-13T13:27:22.338+01:00',
    'type': 'brake'},
```

Listing 3.2: The structure of a turn event.

```
{'category': 'turn',
 'duration': 5.948,
 'end': '2017-12-13T13:27:37.941+01:00',
 'features': {'magnitude': 0.182},
 'magnitude': 0.182,
 'mean': 0.15,
 'peaks': [{'angle': 53.92,
   'duration': 5.948,
   'end': '2017-12-13T13:27:37.941+01:00',
   'is_positive': True,
   'magnitude': 0.182,
   'mean': 0.15,
   'start': '2017-12-13T13:27:31.993+01:00'}],
 'start': '2017-12-13T13:27:31.993+01:00',
 'type': 'left_turn'}
```

In Listing 3.1 and Listing 3.2 both events have some fields that are similar, such as magnitude and integral. However, they also have some fields that differ between them e.g max and angle. To make them more general, one could for example let an acceleration event also have the turn-event specific field 'angle', with value 0.

Listing 3.3: A formatted acceleration event which have the same structure as the other formatted events.

```
{
  'category': 'acceleration',
  'category_int': 0,
  'duration': 12.469,
  'integral': -69.115,
  'latitude': 48.26373,
  'longitude': -4.07703,
  'magnitude': 0.934,
  'max': 4.671,
  'mean': 1.501,
  'angle': 0
}
```

Listing 3.4: A turn event now with the same structure as other formatted events.

```
{
  'category': 'turn',
  'category_int': 4,
  'duration': 5.948,
  'integral': 0,
  'latitude': 48.26373,
  'longitude': -4.07703,
  'magnitude': 0.182,
  'max': 0,
  'mean': 0.15,
  'angle': 53.92
}
```

The general structure was made such that all the event-specific fields were embedded into the general structure with a default value of 0. With each event having the same fields, we now had a similar feature vector across all events making it possible to be used in different machine learning algorithms. An example of two different events with the reformatted structure can be seen in Listing 3.3 and 3.4.

Another part of the reformatting procedure was to append latitude and longitude coordinates to each event. The data containing the events from users did not initially have latitude and longitude coordinates in them, instead the coordinates were stored

in another data offload. One could merge the data containing the events with the latitude and longitude coordinates data, however the problem was that these two data offloads were sampled at different frequencies. For the event data offload, each event the user made was sampled and accounted for. But with the coordinates data, the server only sampled latitude and longitude coordinates every 20 seconds. This means that each driver's position was updated every 20 seconds while sampling event data was occurring continuously. We decided therefore to approximate the location of where each event took place by mapping each event to its closest coordinate data in time. In other words, each event was enriched with latitude and longitude data from the most recent latitude and longitude sample for that specific trip.

3.1.5.2 Filtering users

The amount of data provided by different users varied a lot, this can have a negative effect on the learning of a neural network as in the worst case it will only guess at the class with most data examples. One solution to this which was implemented is to remove users which do not comply with some constrains. A filter on the users was created, removing users and their events if the number of sequences of events was below a certain threshold. One example of a filter could be if a user had less than 1000 event sequences of length 10 they would be removed. This way users that only downloaded the smartphone app and used it once are removed.

3.1.5.3 Normalization

In some multivariate analysis algorithms, it is important to normalize the data in before hand. For example, PCA chooses components that maximizes the variance of the data. Therefore a variable that ranges from 0 to 20 would have a higher impact on the PCA algorithm than another variable that ranges from 0 to 1. Thus, by standardizing each variable prior to PCA we take into consideration that variables scale differently.

For many machine learning algorithms it also helps to normalize data. One example of this is k-means clustering that relies on comparing the euclidean distance between data samples. This means that it tends to create uniform clusters rather than elongated ones. For this reason by not standardizing the data, the variables that take on a higher range of values will impact the algorithm more. It is also beneficial to normalize the data when it comes to neural networks, this makes the network converge faster during training [14].

For these reasons we decided to normalize each floating point feature x_i of the events in the range $[-1, 1]$, as can be seen in Equation 3.1. μ_i denotes the mean and σ_i the standard deviation of each variable.

$$\hat{x}_i = 2\frac{x_i - \mu_i}{\sigma_i} - 1 \tag{3.1}$$

3.2 Machine learning algorithms

Our thesis presents two separate machine learning-based methods, *Latent subspace method* and *Driver norms method*, to obtain a feature vector based on naturalistic smartphone data. This vector will later be used for deriving driver behavior profiles. The former method is a deep learning-based method that utilizes one or more LSTMs to learn both important feature representations and relevant temporal information. The latter method is a method that is based on a drivers average driving characteristic. By analyzing how drivers drive on average, it is possible to derive common patterns for a group of drivers. These two methods are described in more detail below.

3.2.1 Latent subspace method

Here we propose a deep learning method for defining driver behavior for each user based on their trips. This method utilizes a LSTM to extract latent information from sequences of events made by the user. The events consists of accelerations, brakes, turns etc., as well as the magnitude, duration, angle and so on. A sequence of these events are what the LSTM gets as the input, and thus takes into consideration how hard or soft a user accelerates or brakes, as well as sequential information such as if a user normally brakes before a turn or after the turn was made.

3.2.1.1 Main idea

The main idea behind this method is to project the data onto a latent subspace where the drivers, in the subspace, with similar driving behavior are closer to each other. If similarities/dissimilarities are represented well, then a clustering algorithm would cluster together drivers that are close to each other. In other words, it would cluster together drivers that have similar driving characteristics.

One of the reasons why we chose this method is because neural networks learn what features are important for their chosen task, also known as feature extraction. This reduces the amount of domain knowledge needed for tasks such as constructing relevant features for various machine learning algorithms. Furthermore, neural networks excel at mapping its input to an output space where it better represents the data for its task, thus by trying to teach a neural network which driver is driving during a certain trip. It will learn a better latent subspace for expressing driver behavior. We decided therefore to use the *user_id* field in the data as target label for the neural network. By using *user_id* as the classes to learn and predict, the network tries to learn patterns in the data that distinguishes the drivers. If two drivers drive similarly however, the network will have a hard time telling them apart and they would thus be close to each other in the latent subspace.

Figure 3.6 shows an overview of the proposed method. As the figure shows, the method can be described in two steps. The first step is to train a classifier with the *user_id* of different drivers as target labels. The real purpose of the first step

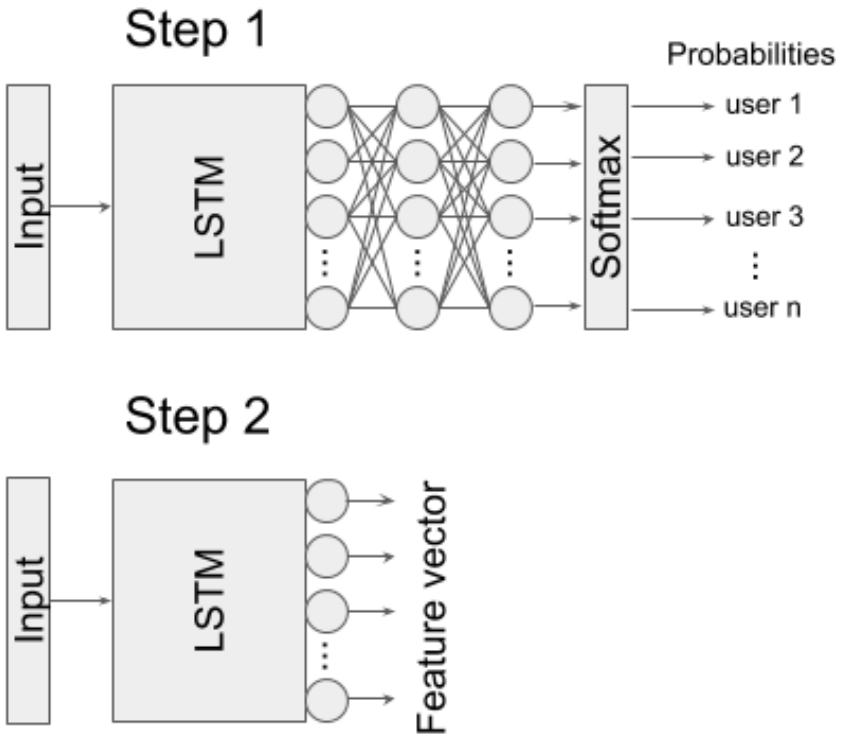


Figure 3.6: An overview of the proposed *Latent subspace method*. *Step 1* shows the supervised training process of the network. Later in *step 2* a feature vector is extracted for each driver from an intermediate layer in the network when forward propagating sequences of events. These feature vectors are known as the latent subspaces of the drivers.

is not to actually build a driver classifier, but to get a good representation of each driver's behavior pattern in a latent subspace. Thus the first step is supervised and its main purpose is to extract latent information about each driver's driving style. The second step is done after the classifier has been trained and when we have a feature vector representing each driver's driving style. This step is basically to cut off the last layer with the softmax output and extract the feature vector of one of the last fully connected layers. Thus we will have a high-level representation of each driver's driving pattern. This feature vector or latent subspace will then later be used in combination with an unsupervised method to cluster similar drivers. In summary, this method utilizes both supervised and unsupervised learning to extract and cluster similar driving styles for different drivers.

3.2.1.2 Creating the network architecture

Throughout the thesis work we tried different LSTM architectures. The following architectures were implemented:

Single LSTM net
The first type of network architecture built was based on one LSTM layer with a varying amount of fully connected layers on top of the LSTM before the classifier.

The input to the network was a matrix with three dimensions, batch size, sequence length and number of features. The output from the classifier was the probability of each user. The network architecture is shown in Figure 3.7.

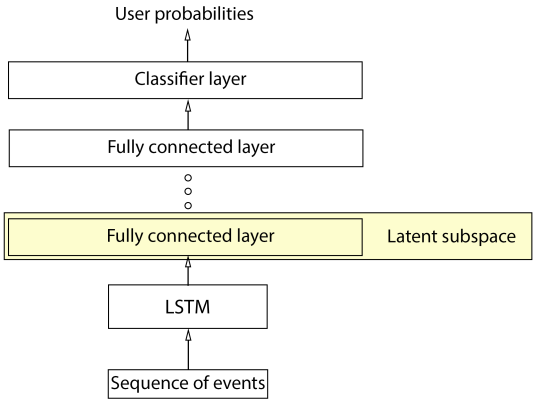


Figure 3.7: A schematic overview of the network architecture of *Single LSTM net*.

Multiple LSTM net
The later usage of the network was to separate the users on their overall behavior and not on specific sequences of events. Therefore we wanted the network to focus less on individually sequences. This led to the idea of using multiple LSTMs and averaging the results of them as input to the rest of the network. Therefore the second architecture built used multiple parallel LSTM layers. The LSTM layers shared weights and the outputs from them were averaged and fed to a varying amount of fully connected layers as well as in the single LSTM net. The input to the network had four dimensions, *batch size*, *lstm layer*, *sequence length* and *number of features*. An overview of the architecture is shown in Figure 3.8.

3.2.1.3 Forming input feature vector

Since we used an LSTM, a sequence of events is used as input, which also means that we had to do some additional pre-processing to get the desired feature vector for the network. LSTMs require sequences of inputs, thus we had to decide on how many number of events a sequence would consist of. We did not know the appropriate amount of events that should have been in each sequence and decided therefore to experiment with different sequence lengths. If one uses a sequence length that is too short, then it will be hard for the network to find any long term dependencies in the driving pattern. But if one were to use a sequence length that was too long, then the network would more or less learn the road topology instead of the actual driving behavior.

We decided to try and limit the sequence length so that we only input events that corresponds to values in the range of 10-60 seconds of driving.
In Figure 3.9 we can see a plot of the frequency of the difference in time between two events in succession. The majority of succeeding events have a small time window between them, this is reasonable as a driver could, e.g, turn and accelerate at

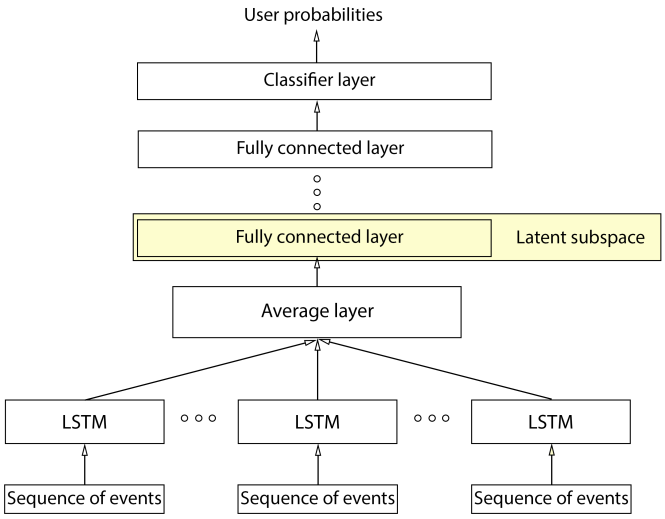


Figure 3.8: A schematic overview of the network architecture of *Multiple LSTM net*. The average layer averages the inputs from the LSTMs, thus an average behavior over time is obtained.

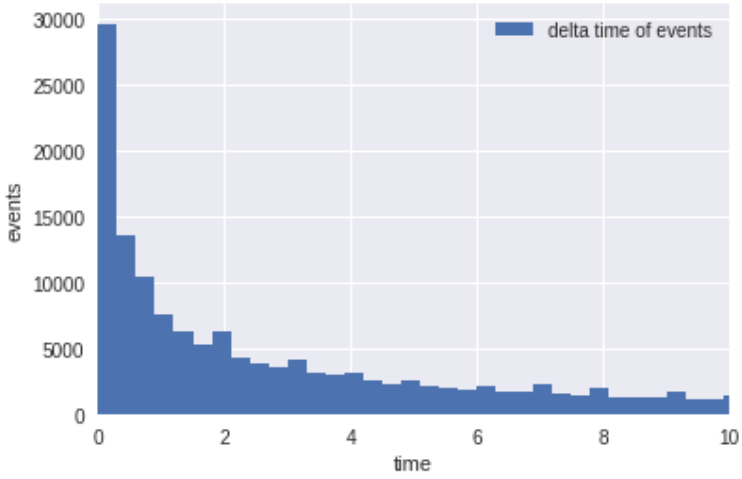


Figure 3.9: The frequency of different times between events.

the same time. The mean of difference between two succeeding events was approximately 2 seconds, we decided therefore to start with a sequence length of 10 events. With a sequence length of 10 events we would use input sequences that encoded 20-30 seconds of data.

For the *Average LSTM net* architecture we needed one more sequence of events in the input for each additional LSTM layer added to the network. The optimal approach would be to divide the number of samples per user evenly on each LSTM

layer. This was not feasible as the number of samples per class would become too small. When training on 395 users for example, the user with the smallest amount of samples would have 1000 samples, which is feasible to train one LSTM layer, but dividing that data on five or ten LSTM layers would not yield enough samples per class to train the network. Therefore the approach adopted, which is shown in Figure 3.10, was a sliding window approach. By using a window size of *sequence length* and a stride of 2, new sequences of events were generated from events originating from the same trip for each user.

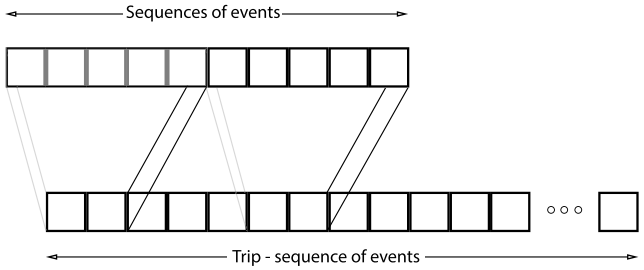


Figure 3.10: The sliding window method used for generating data for the *Average LSTM net*, here with a window size of 5 and a stride of 2. In the first generated sequence the last three events are shared with the second generated sequence.

3.2.1.4 Visualizing latent subspace

We want to visualize the latent subspace for analysis and see if distinct clusters exist. But depending on how many neurons one uses at each layer, the corresponding latent subspace for each layer will have as many dimensions as the number of neurons that layer has. Therefore we used PCA and t-SNE for visualization since these dimensionality reduction techniques can reduce the number of dimensions to 2 or 3.

The layer that we are interested in visualizing is the layer directly after the LSTM. This is because we use fewer neurons on that layer which improves PCA and T-SNE approximations, also later layers will try to project the data onto a more linearly separable space, making clusters less distinct.

3.2.2 Driver norms method

Our data consists of many trips done by various drivers. Each trip is a sequence of events done by that corresponding driver, which means that we have the possibility of analyzing how drivers behave in certain situations and how the values of their events differ or conform with other drivers. Another proposed method is therefore to derive driver behavior by establishing certain similarities and/or differences between drivers based on how they act during different events.

3.2.2.1 Main idea

The main idea behind this method is to first extract an average behavior during certain events for each driver by averaging his or her event values. The reasoning behind this is that by averaging the values for each event, we get an indication of how each driver normally e.g accelerates or turns. There will naturally always be some outliers for each driver, but we believe an average of all event data is a good enough approximation.

The second step, after we have established an average behavior/pattern for each driver, is to group together drivers with similar driving patterns. If one driver is more prone to speeding, in comparison to other drivers, then his average acceleration magnitude would be higher than the rest of the drivers. We will group together drivers by utilizing clustering algorithms.

3.2.2.2 Feature selection

The feature selection process for clustering on user events began by selecting a subset of features from the already existing ones. This is done to reduce the impact of noisy or unimportant features on the clustering algorithm.

By checking the correlation between features, one can see and remove features that correlate too much. This is beneficial since e.g by using two features that correlate to a great extent, you are not adding much new information in comparison to just using one of them. In addition, you are also increasing the number of dimensions and complexity of the data.

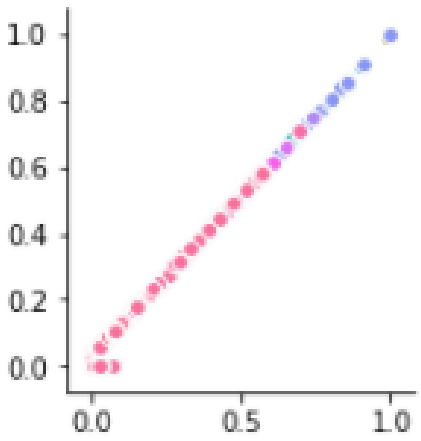


Figure 3.11: The correlation between the magnitude and the max value for acceleration events. The linear shape of the data in the graph indicates that these two variables correlate to almost 100%.

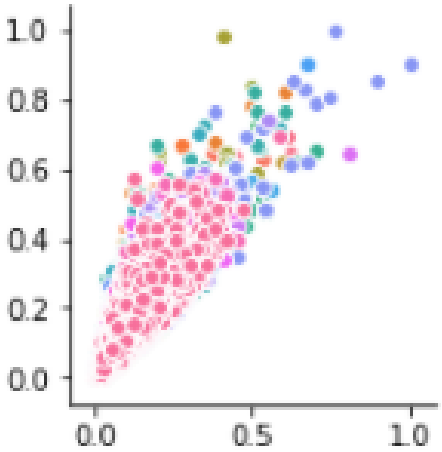


Figure 3.12: The correlation between the magnitude and the mean value of acceleration events. The graph indicates that these two variables are highly correlated.

We checked the correlation between all variables for 100 users to see which variables were redundant. In Figure 3.11 and in Figure 3.12 we can see three variables that are highly correlated. This means that using these three variables together does not add much more information in comparison to just using the magnitude of acceleration events. We decided therefore to drop the mean and max value of acceleration events for this method. The full extensive graph over feature correlations can be found in appendix A

Another feature selection process was removing features that did not add valuable information regarding driving behavior. This is done based either on domain knowledge or intuition. For turn events we removed the angle of the turn since the turn angle says more about the road topology than the actual driving behavior itself. An example of this would be if one driver had an average turn angle of 20 degrees and another driver had an average of 0 degrees, then this could mean that the former driver drives more in inner cities whilst the latter driver drives more on highways. For this reason we decided that the turn angle was insignificant with respect to driving behavior.

3.3 Clustering

We proposed earlier two methods to derive feature vectors that represent driver behavior, namely *Latent subspace* and *driver norms*. These two methods will produce feature vectors that we will cluster on, this will group together drivers with similar driving patterns. The workflow from inputting pre-proccesed data into the two methods and clustering on the output is shown in Figure 3.13. The clustering algorithms used throughout the thesis work were k-means clustering, DBSCAN and Agglomerative clustering. We used these three and compared the results to see which algorithm made more sense for our data.

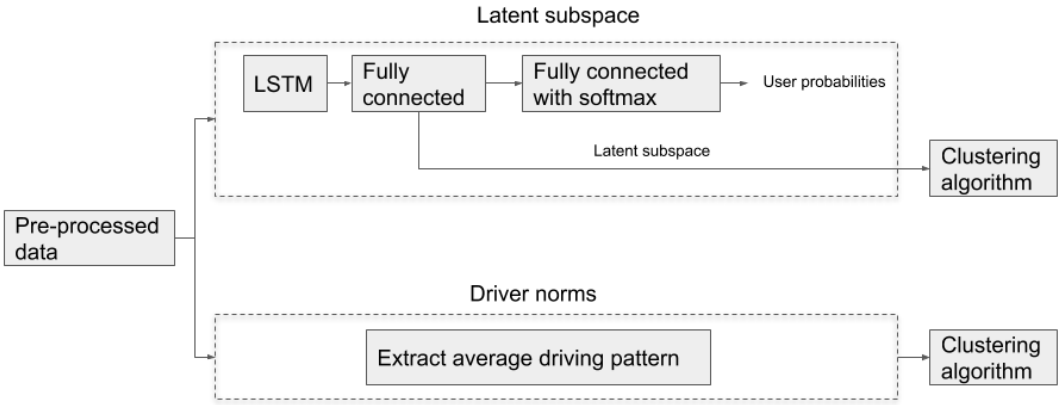


Figure 3.13: The workflow for clustering on the drivers. The feature vector to cluster on is derived from the two proposed methods in this thesis.

3.3.1 Interpreting clusters

Trying to interpret the clustering results from our proposed methods is the first step to understanding if there exists distinct driver profiles and if so, also try to understand what they are. From the results we will be analyzing and comparing the distributions of the different events to see how the different clusters differ from each other. We will also be looking at the mean and variance of each cluster, since they are two important statistical features describing a distribution, and draw conclusions regarding the different clusters average pattern and deviation in that specific pattern. Lastly, we will also be comparing the different method's clusters and see how the methods vary from each other in regards to producing clusters.

3.3.2 Evaluation

The two methods proposed, latent subspace clustering and clustering on driver norms, will be applied and tested on pre-processed data. The tests will first be done on a small sample of users to see how the methods works for a small sample. Afterwards, the methods will be tested on a bigger sample. To measure and evaluate cluster quality, we are going to use two methods for measuring inter-cluster similarities and intra-cluster dissimilarities. The following methods are going to be used

Silhouette score
The first evaluation criteria is going to be to measure the silhouette score for all points in each cluster. We will then average the values over all clusters to obtain an average silhouette score. A high score is more preferable here.

Elbow method
We are going to use the elbow method from $k = \{2, 3 \ldots, 9\}$ to measure the average sum of squared error for the clusters. The point for an optimal number of clusters is going to be determined by finding the elbow point. The elbow point cannot always be unambiguously found. But for cases when the elbow point is clear, we will take that into consideration along with the silhouette score.

4
Results and Evaluation

This chapter will present the different experiment results from the two machine learning algorithms explained in Sections 3.2.1 and 3.2.2, namely *Latent subspace method* and *Driver norms method*. The experiments concerning the former method will be presented in Section 4.2 and experiments from the latter method will be presented in Section 4.3. Before this Section 4.1 will first present the setup and what software frameworks were used.

4.1 Computer and software libraries

The software frameworks used during the project were Tensorflow, Keras and Scikit-learn.

Tensorflow is an open-source machine learning library that allows user to construct data flow graphs [33]. Each node in the data flow graph is responsible for a mathematical operation, using these in sequence allows the user to easily build different machine learning algorithms and neural networks.

Keras is an open-source high-level machine learning library that runs on top of a low-level machine learning library such as Tensorflow [34]. The purpose of Keras is to allow more rapid building and experimenting of neural networks.

Lastly, Scikit learn is a commercially used open source library that presents simple tools for data mining and data analysis [35]. This simplifies tasks such as data visualization, dimensionality reduction and evaluation of machine learning results.

All the experiments were conducted on a Linux computer with dual Intel Xeon CPUs and 32Gb RAM.

4.2 Latent subspace experiments

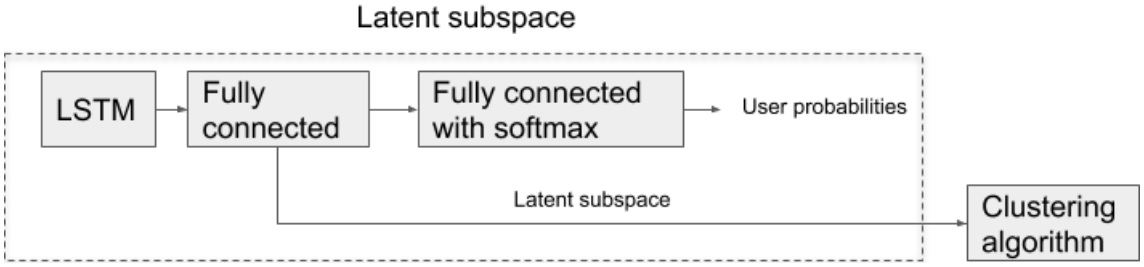


Figure 4.1: The structure of Section 4.2 follows the structure of the figure. First the training of the LSTM and neural network is described in Sections 4.2.1, 4.2.2 and 4.2.3. Then clustering on the latent subspace is described in Section 4.2.4.

In this section the experiment results using the latent subspace method described in Section 3.2.1 will be presented. Following the structure of Figure 4.1, first a brief description of the training and hyperparameter search of the network are given in Section 4.2.1, then the results of the hyperparameter search are given in Section 4.2.2. Also an evaluation on the test dataset is given in Section 4.2.3.

As can be seen in Figure 4.1 the next step of the method is the latent subspace clustering which is described in Section 4.2.4. There three experiments will be presented and the clustering of them.

4.2.1 Training the neural network

When training the network different hyperparameters were used. We experimented with different number of neurons and different number of layers in the network. Other parameters were also if additional data was to be used or not, and the number of users to train on. During early experiments with hyperparameters concerning the network structure after the LSTM layer, i.e. the fully connected layers, no significant improvement was found. Therefore the hyperparameter search for the network was mainly focused on the hyperparameters presented in Table 4.1 below. The train, validation and test split of the different datasets was 75% train, 19% validation and 6% test.

4.2.2 Hyperparameter search for the Neural Network

This section presents the results of the hyperparameter search concerning the *additional data* and the number of *users* presented in Table 4.1. Different number of *LSTM layers* were also tested, but only networks with 10 layers will be presented, as these gave higher accuracy. In Figure 4.2 the top-5 accuracy for net 1 and 3 is presented. Respectively Figure 4.3 show the top-5 accuracy for net 5 and 7. In both the case of 100 and 395 users there is a significant boost in accuracy when adding

Table 4.1: The different hyperparameters for the different network architectures tested in the first parameter search.

Net	LSTM layers	Additional data	Users
1	10	no	100
3	10	yes	100
5	10	no	395
7	10	yes	395

Table 4.2: The top-5 accuracy when evaluation on the test data set.

Net	top-5 accuracy
1	22.0%
3	31.1%
5	21.8%
7	44.5%

the additional data. The networks trained using 100 users have better accuracy than those with 395 as there are fewer classes. However it is not certain that networks on 100 users will perform a better behavior clustering than those based on 395 users as the latter ones can potentially map more behaviors.

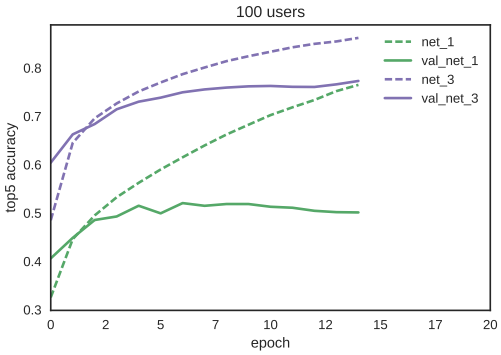


Figure 4.2: The top-5 accuracy of net 1 and 3. The doted line is the train accuracy and the continuous line is the validation accuracy.

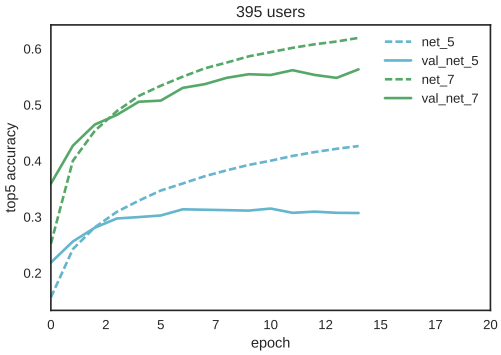


Figure 4.3: The top-5 accuracy of net 5 and 7. The doted line is the train accuracy and the continuous line is the validation accuracy.

4.2.3 Evaluation on Test Data

Evaluation on the test dataset showed the same trend as in the hyperparameter search shown in Figures 4.2 and 4.3. Architectures with external data perform better in both the case of 100 and 395 *users* as can be seen in Table 4.2.

4.2.4 Latent subspace clustering

Three network architectures will be presented here, *net 3* with 100 users, *net 5* and *net 7* with 395 users. The difference between the last two mentioned nets is that the former one is not trained with additional data, while the latter one is. Clustering was performed using the k-means algorithm on the latent subspace from the first hidden layer.
Because network *net 0*, *net 1* and *net 2* produced similar clusters as *net 5* and since network *net 4* and *net 6* gave similar clusters as *net 5* and *net 7*, we chose to only further pursue *net 3*, *5* and *7*.

4.2.4.1 Net 3

The sum of squared errors for *net 3*, which can be seen in Figure 4.5, suggests a clustering with $k = 3$. However, the silhouette score from Figure 4.4 suggests a clustering with $k = 2$. We chose to use three clusters so we could analyze the dissimilarities a bit more, a t-SNE projection of these three clusters can be seen in Figure 4.6.

Figure 4.4: The silhouette score for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 3*.

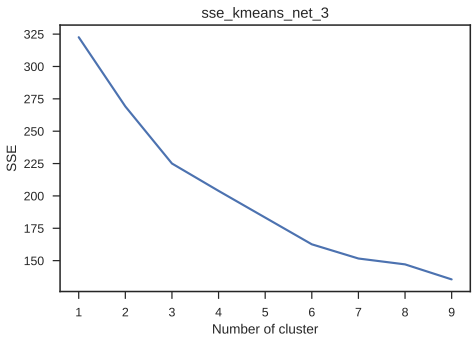


Figure 4.5: The sum of squared errors for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 3*.

The histograms in Appendix B.5 show the mean of each turn event feature for each user in the three different clusters suggested by the k-means clustering. Appendix B.1 shows histograms containing equivalent information, but for the acceleration events instead. Cluster 2 is the cluster that stands out the most if one were to analyze the histograms. One can see that they tend to have a lower overall acceleration and brake integral, which indicates a more cautious tendency. However, for mid speeds they have a higher integral than the rest of the population as well as a much higher turn magnitude. This might suggest that they are more aggressive inner city drivers. Cluster 0 and 1 are however harder to interpret since judging by their histograms, they are very similar for most scenarios. Although they have some small differences when it comes to high speed areas, cluster 0 prefers accelerations

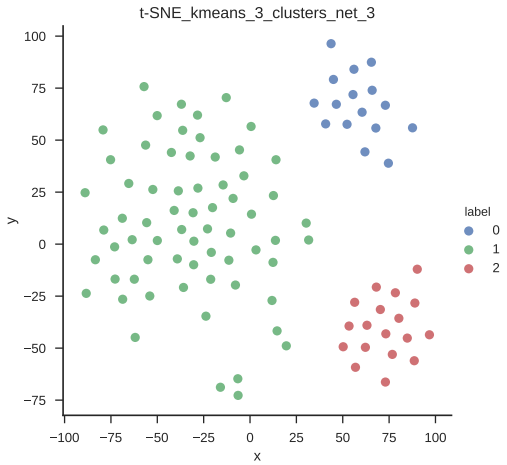


Figure 4.6: A t-SNE projection of the k-means clustering with $k = 3$ on the latent subspace of net 3.

with a lower mean value and a higher duration, and cluster 1 is the opposite with a lower duration but a higher mean acceleration.

The different clusters are visualized using Kibana and as can be seen through Figures 4.7 to 4.9, there exists some geographical correlation between the drivers in each cluster. Cluster 0's drivers are mainly based in the southeast of Europe while cluster 2's drivers are mainly based in Asia.

In summary, for net 3 we conclude that cluster 2 suggests some sort of aggressive inner city driving. Cluster 0 and 1 are very similar, however the drivers in cluster 0 are more prone to having higher acceleration and brake integrals on high speed roads. Also, cluster 0 are more prone to be having a lower mean acceleration with a higher duration while cluster 1 is the opposite, meaning that they prefer accelerating harder with a lower duration.

Figure 4.7: Drivers in cluster 0's geographical location for *net 3* using Kibana.

Figure 4.8: Drivers in cluster 1's geographical location for *net 3* using Kibana.

Figure 4.9: Drivers in cluster 2's geographical location for *net 3* using Kibana.

4.2.4.2 Net 5

For *net 5* both the silhouette score and the sum of squared errors suggests a clustering with $k = 3$. The t-SNE projection for *net 5* in Figure 4.12 displays the clusters for the given value of k.
The histograms in Figure B.6 in Appendix B show the mean of each turn event feature for each user in the three different clusters from the k-means clustering. Figure B.2 shows histograms containing the same information, for the acceleration events.

As can be seen in Figure B.2, the acceleration mean feature of cluster 0 and 2 are a bit higher than those of cluster 1, together with the duration feature which is a bit lower on average for cluster 0 and 2 this suggest that cluster 0 and 2 may be aggressive. Figure B.6 show distinct difference in both the duration and the magnitude for cluster 0, where the duration is distinctly shorter and the magnitude is distinctly higher. This is suggests that cluster 0 map an aggressive behavior.
The Kibana visualization of the three clusters can be seen in Figures 4.13 to 4.15. No cluster is confined to a particular geographical area, this together with the fact that no external data was used when training the *net 5* shows that the aggressive driving behavior is not confined to a particular geographical area.

In Figures 4.13, 4.14 and 4.15 a geographical map is shown displaying cluster 0's, cluster 1's and cluster 3's drivers' location. The map is showing hot spots over the location of each driver's events. These two figures show that the three clusters have no distinct difference in geographical location.

Figure 4.10: The silhouette score for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 5*.

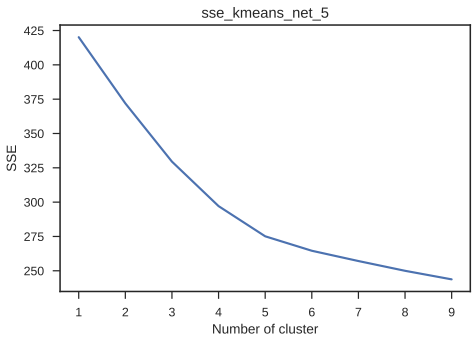


Figure 4.11: The sum of squared errors for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 5*.

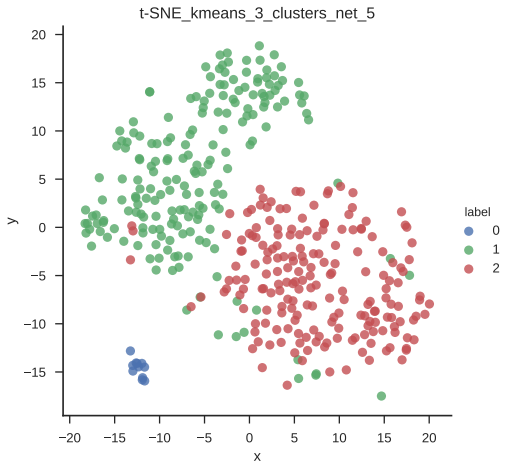


Figure 4.12: A t-SNE projection of the k-means clustering with $k = 3$ on the latent subspace of *net 5*.

Figure 4.13: Drivers in cluster 0's geographical location for *net 5* using Kibana.

Figure 4.14: Drivers in cluster 1's geographical location for *net 5* using Kibana.

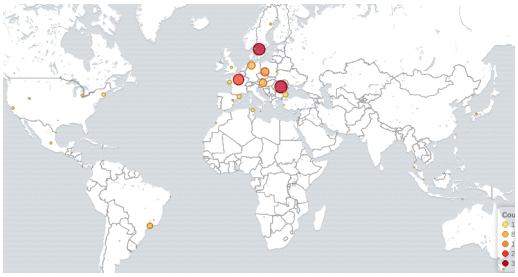


Figure 4.15: Drivers in cluster 2's geographical location for *net 5* using Kibana.

4.2.4.3 Net 7

The silhouette score for *net 7* peaks for $k = 5$, which can be seen in Figure 4.16. The sum of squared errors does not give a distinct elbow point but suggests a k value of 4 or 5, taking this in consideration with the silhouette score suggests that $k = 5$ is an appropriate amount of clusters.

Figure 4.16: The silhouette score for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 7*.

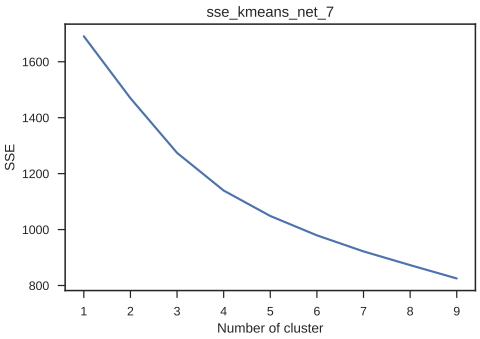


Figure 4.17: The sum of squared errors for different values of k in the k-means clustering algorithm when clustering on the latent subspace of *net 7*.

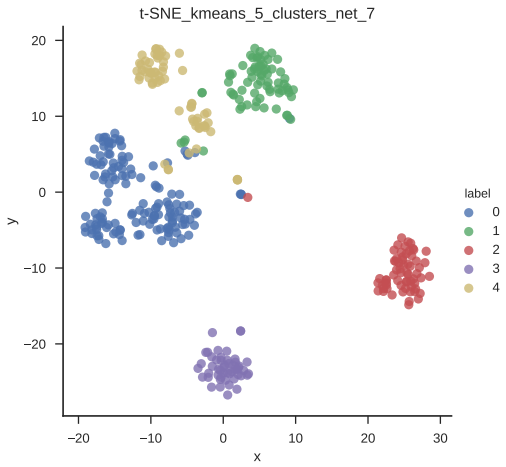


Figure 4.18: A t-SNE projection of the k-means clustering with $k = 5$ on the latent subspace of net 7.

The histograms in Appendix B.7 show the mean of each turn event feature for each user in the five different clusters suggested by the k-means clustering. Appendix B.3 shows histograms containing equivalent information, but for the acceleration events instead.

As can be seen in the figure in Appendix B.3 users in cluster 3 have a higher mean accelerations than other users. From this figure it is also clear that drivers from

cluster 1 have higher duration as well as a higher integral. This does not have to be a sign of aggressive behavior but cluster 3 is more likely to map an aggressive behavior due to the high acceleration means. Analyzing the turn events reveal that cluster 2 have higher magnitude in turn events filtered on mid speed, which can be seen in Appendix E.14. This is also true for cluster 3 when filtering on low speed areas which is illustrated in Appendix E.13.

The Kibana visualizations reveal that some clusters map to certain geographical areas which might be a sign of overfitting to the open street map data. Figure E.20 show the Kibana visualization for cluster 3 which have most of the events centered around the southeast of Europe.

Figure 4.19: Cluster 3's driver's geographical location for *net 7* using Kibana.

4.3 Driver norms experiments

Our other proposed method, clustering on driver norms, was used on both data without external data and on data combined with external data. The reason behind applying it on the two different datasets is to determine how external data affects the clustering and what the difference in results is. The clustering on driver norms along with a visual representation of the different clusters will be presented here for the two datasets.

4.3.1 Without additional data

The sum of squared errors and the silhouette score for clustering on driver norms without external data are shown in Figures 4.20 and 4.21. The silhouette score indicates that, in this case, $k = 3$ is an optimal value for k to maximize intra-cluster similarities and inter-cluster dissimilarities. However, the figure displaying the sum of squared errors does not have any clear signs of what the optimal value for k should be. Since the elbow point cannot always be unambiguously chosen, we use $k = 3$ which is the recommendation from the silhouette method. The visualization of the corresponding clusters is found in Figure 4.22.

The algorithm used for the clustering was Agglomerative clustering with $k = 3$. The visualization of the three different clusters shows that the projection of the data onto

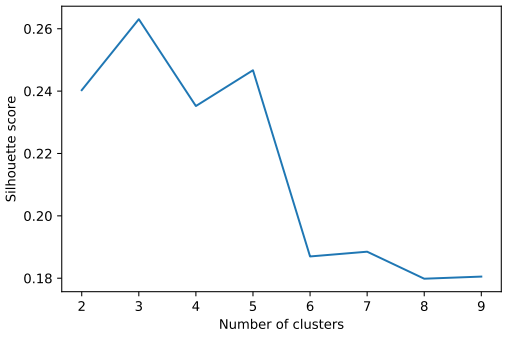


Figure 4.20: Silhouette score for the dataset without external data.

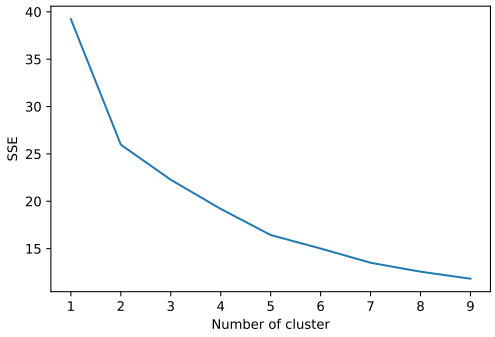


Figure 4.21: The sum of squared errors for the dataset without external data.

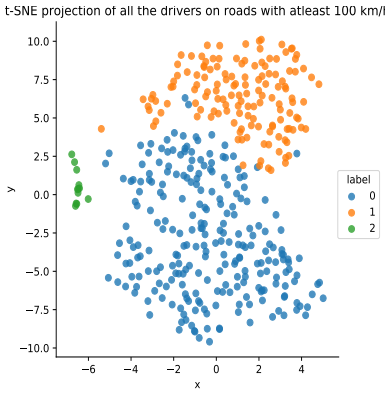


Figure 4.22: The three different clusters obtained from Agglomerative clustering with $k = 3$. The data is projected down to two dimension with t-SNE.

a two dimensional space results in an elongated shape with a small cluster on the outside. The intuitive interpretation of this clustering result is that some drivers are well represented in their cluster, and some drivers are somewhat in between the two largest clusters. This is reasonable since there will always exist drivers that are in between different driving profiles.

As seen in Figure 4.22, three clusters were obtained by running agglomerative clustering on the driver norms. It is apparent that most of the drivers belong to either class 0 or class 1, while there are very few drivers in cluster 2. An explanation for this is that the drivers in cluster 2 most likely have something in common that is not typical for many of the drivers. The most notable distribution plots for these clusters is presented in Figure 4.23 to 4.25 which gives a better understanding of what distinguishes these groups of drivers.

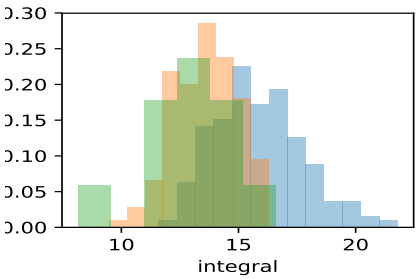


Figure 4.23: The clustering on driver norm's distribution plots for acceleration events.

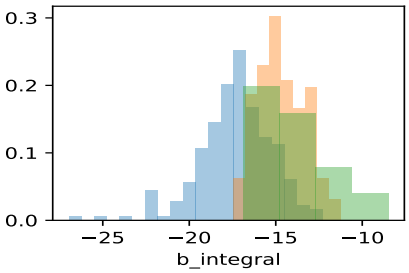
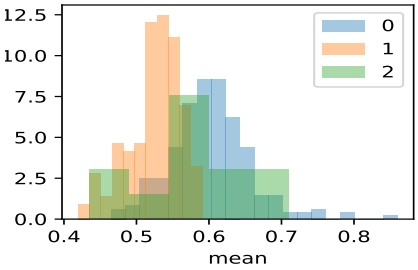
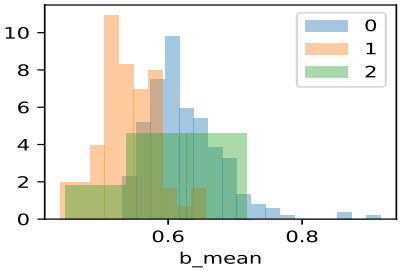


Figure 4.24: The clustering on driver norm's distribution plots for brake events.





As seen in the above figures, the difference between cluster 0 and cluster 1 is the integral and mean of both acceleration and brake events. Since a higher integral means an overall higher speed, one can argue that cluster 1 are more cautious drivers and cluster 0 are more normal/speedy drivers. Cluster 2, the small group of outliers in Figure 4.22, show very distinct behavior when it comes to turn events. They are on the very low end of the spectrum when it comes to turn duration and on the very high end of the spectrum for turn magnitudes, which can be derived from 4.25. They show however very similar behavior to both cluster 1 and cluster 0 for acceleration and brake events. In other words, they are very similar to both cluster 0 and 1 for acceleration and brake events, but they are very different in terms of turn events where they tend to go in/out from turns at a higher speed. This might indicate some sort of aggressive driving for cluster 2. In summary, one could view these three clusters as groups of drivers showing cautious/normal, normal/speedy and aggressive behavior.

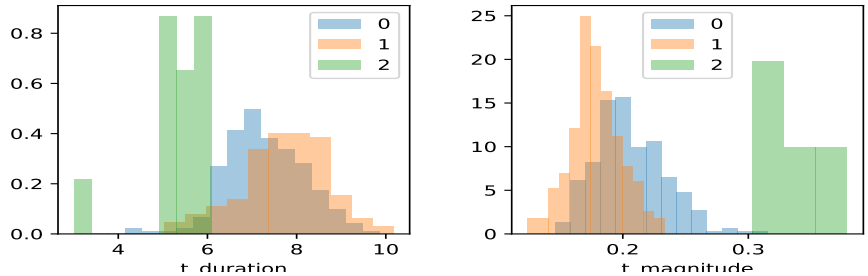


Figure 4.25: The distribution plot for turn events, for clustering on driver norms

4.3.2 With additional data

By combining the original dataset with additional data, we could more specific and thorough in our analysis. Through the additional data we were able to utilize information such as maximum speed limits on roads, lighting conditions etc. This made it possible to analyze and cluster users based on specific scenarios. For our analysis with the additional data we chose to analyze drivers on three different road types, namely *low speed*, *mid speed* and *high speed* roads. We chose to define these three road types as follows:

1. *Low speed roads*: roads with a speed limit in the range of 0-30 km/h
2. *Mid speed roads*: roads with a speed limit in the range of 50-80 km/h
3. *High speed roads*: roads with a speed limit of at least 100 km/h

In Figure 4.26 we see the sum of squared errors for roads with 0-30 km/h speed limits. By using the elbow method, this graph indicates 3 is a reasonable number for k. By using $k = 3$, the following three different clustering results were obtained for the different speed limits.

The above figures shows us three different clusterings on three different road types. It is important to note that these three clusterings are different from each other,

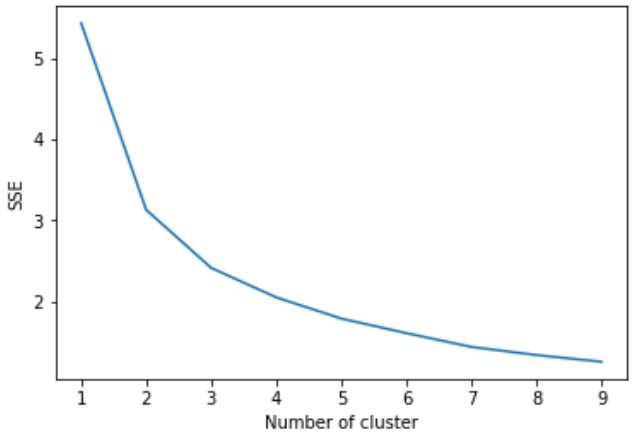


Figure 4.26: The sum of squared error (SSE) for the dataset with external data.

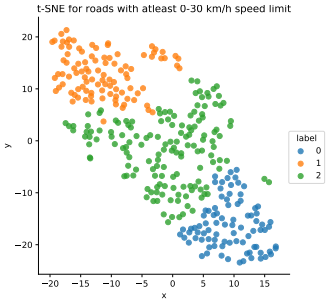


Figure 4.27: A t-SNE plot for roads with 0-30 km/h speed limit. Each point represents a user.

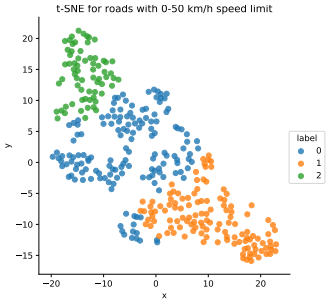


Figure 4.28: A t-SNE plot for roads with 30-50 km/h speed limit. Each point represents a user.

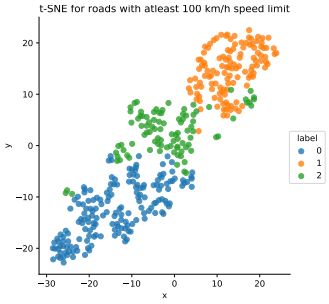


Figure 4.29: A t-SNE plot for roads with atleast a 100 km/h speed limit. Each point represents a user.

e.g class 0 for *low speed* areas is different from class 0 in *mid speed* areas. In other words, these three different clusterings are individual scenarios and should be viewed as three different non-related results. The idea behind clustering on different road types is that we can analyze and interpret the clusters more specifically for each scenario since there are fewer external factors to take into account.

To get some sort of interpretation of the clusters we analyzed the acceleration, brake and turn events for each cluster. The most prominent ones are displayed in the figures 4.30, 4.31 and 4.32 below

The graphs show the distribution of values for the three different clusters. There are clear differences between them in terms of acceleration and braking, this is true for all three ranges of speed limits.

We can see that the three different clusters represent some sort of 'below average', 'average' and 'above average' group, meaning that the 'average' represents some sort

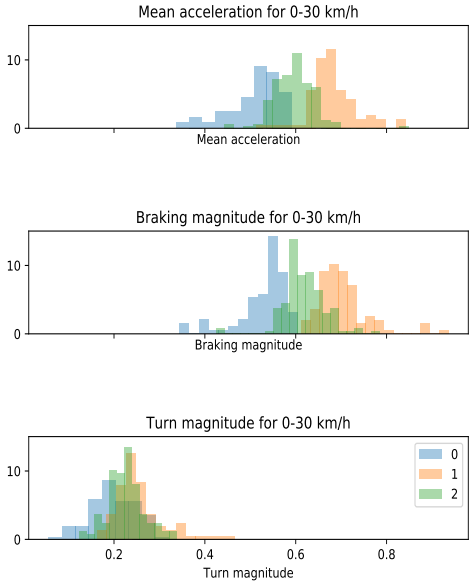


Figure 4.30: The histograms for roads with a speed limit in the range of 0-30 km/h.

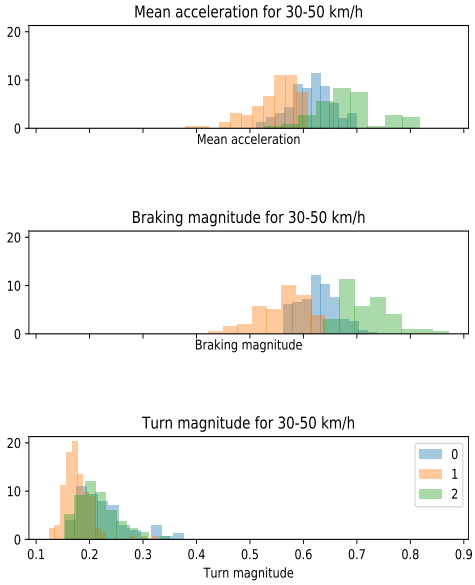


Figure 4.31: The distribution plot for roads with a speed limit in the range of 30-50 km/h.

of driving norm for the corresponding speed limit. This would then also mean that the 'below average' group have means and magnitudes that are lower than the norm, and vice versa for the 'above average' group. Another interesting aspect that one can see from the graphs is that drivers that have a higher acceleration also have a higher braking magnitude, the same thing applies for lower acceleration and braking magnitudes. This is reasonable since a higher acceleration would mean a higher driving speed, which in turn requires harder brakes in different situations where one would need to slow down.

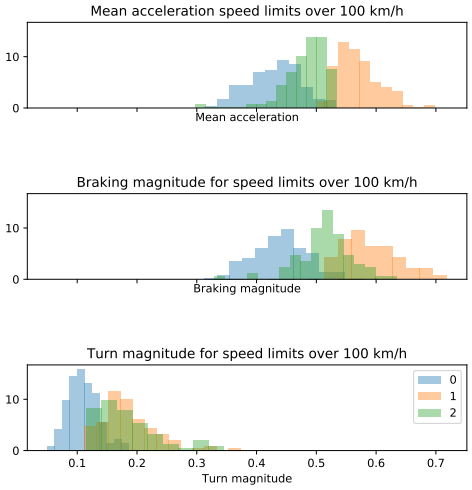


Figure 4.32: The histograms for roads with a speed limit of at least 100 km/h.

5
Discussion

In this section we will note and discuss important elements regarding the project. It will mainly be about the data, the methods and the outcome of the thesis work.

5.1 Results

We will now cover a discussion regarding the results down below.

5.1.1 Latent subspace clustering

The results for this proposed method varied a lot depending on if one were to add additional data or not. For *Net 5*, which was the network trained without additional data, the results were very similar to the ones derived from the driver norms method. Both our evaluation methods indicated that three clusters were optimal which was the same for the driver norms. Also, the significance of the different clusters were similar across both methods. The clusters differed mostly on acceleration integral, brake integral and on turn magnitude for the different drivers. Since the data consisted mostly of these events, it might be hard for the algorithm to find similarities or dissimilarities regarding other features.

For the results on *Net 7*, which was the network trained with additional data, the results differed a lot from *Net 5*. The visualization of the clusters show that the clusters are much more distinct than for the results without additional data. Considering the fact that the additional data adds more contextual information regarding each drivers trip. This might indicate that the network instead learns driver that drives under similar contextual conditions. For instance, it might group together drivers that drive a lot on highways or drivers that drive in inner cities a lot and so on. This hypothesis is supported the Kibana plots for *Net 7*, which is found in Figures E.17 to E.21 in the appendix. One can see that there exists a geographical correlation between the drivers in each cluster. For example drivers in cluster 0 are mostly based in Europe while drivers from cluster 1 are mostly based in the United States and so forth. This is important to keep in mind while evaluating the clustering results for this method with additional data.

Since a big part of LSTMs is to take into account the temporal information of each event, some of the less obvious clusters might have been a result of the temporal information behind their events. Interpreting these kind of things or seeing exactly

what type of sequences are common for a cluster is a very hard task, but still a possible one. Lately, attention mechanisms have been used widely in the field of natural language processing (NLP) to see what inputs trigger certain outputs. For instance, in a translation task one could utilize this attention mechanism to see which words contributed the most to the translated output. An idea here could be to implement some sort of attention mechanism to see which sequences of input events contributes the most to the different outputs. Then one could potentially derive certain common sequential traits among a group of drivers, such as they might be prone to be braking before a turn in comparison to some other group that might be more prone to be braking during the turn and so on. A big part of driver behavior is the temporal information in the events, however this is also the hardest part to analyze and understand. This area is therefore definitely a suggestion for future research for others who might be interested in analyzing these kinds of behavior.

5.1.2 Clustering on driver norms

All in all, by clustering users based on a combination of driving norms for each user and external data such as speed limit for different roads, we are able to find behavioral patterns in terms of acceleration and braking magnitudes. From the distribution graphs presented, one could see that drivers that accelerated more also were prone to be braking more or having a higher braking magnitude at least. However, also judging by the distribution plots, higher acceleration and braking events did not really have an impact on turn events. One could think that a higher overall acceleration and braking magnitude would indicate an aggressive driver, and it would be reasonable for aggressive drivers to also have a higher turn magnitude. This is not the case which raises some questions. One probable reason could be that the difference between a 'normal' or 'average' turn event and an aggressive one, in terms of turn magnitude, is so small that it is hard to distinguish them.

The added external data helped us to narrow down the different events into similar driving scenarios or conditions instead, this means that we could analyze driver norms more thoroughly and with more precision. When analyzing driver norms without external data, certain aspects such as road type and speed limit is not taken into consideration. This means that if two users have significant differences in e.g acceleration magnitude, then it could maybe be because one of the driver drives more in residential areas whilst the other drives more on highways. So even if two drivers have the same pattern for acceleration and braking events under similar conditions, the fact that if they differ a lot in regards to road type they usually drive on would mean that their acceleration and braking magnitudes would differ a lot as well. Therefore by not adding external data, the analysis would be too general to really derive anything of significance. This does not mean that adding external data makes the analysis super accurate, it does however take other external factors into consideration which potentially could have a big impact on a user's driving behavior, thus giving a result that is more thorough and probable.

5.2 Methods

5.2.1 Latent subspace clustering

How well the latent subspace represents driver behavior largely depends on the neural network's ability to distinguish the drivers. That is why a higher performing network is desirable, however we deem that a good top-5 accuracy is sufficient instead of trying to get a high top-1 accuracy. The reason for this is that the task of predicting 395 different users is going to have a really low baseline accuracy if one were to randomly guess the driver. Also, in our opinion it does not make a huge difference if the right driver for a certain trip was the second or third guess by the network. This is still an indication that it has learned relevant features, as well as that the latent subspace has a decent representation of different user's driver behavior. With this said, one should still strive to get the best possible performing network but without getting too caught up on trying to get the highest top-1 accuracy in our opinion.

As far as the different network architectures are concerned, the best performing ones were the ones that utilized multiple LSTMs and averaged their output value. We are not surprised since inputting multiple trip segments and averaging the different LSTMs output values would yield some output with much less noise than just trying to guess the right driver from one input.

5.2.2 Driver norms

This method utilizes a more traditional approach, which is to try and construct a relevant feature vector for driver behavior and then utilize appropriate clustering algorithms to group similar drivers. The benefits of this method is that its relatively fast, at least in comparison to approaches with neural networks, and it's output is also relatively easy to interpret. However, it has quite a few drawbacks. One of the biggest drawbacks is that it requires quite an amount of domain knowledge to really extract and construct good features for representing drivers. A suggestion for future research would be to try and engineer new features that are relevant to do driver behavior. Since the data we clustered on can be considered to be on a low level, meaning that it only consists of magnitudes, durations and so on, the clustering results will be confined to driver profiles based on magnitudes, durations etc. Other feature representations for drivers that are more informative could potentially both give better clusters, as well as easier interpretations of different driver behaviors.

5.3 Data

The data is the most important aspect when doing statistical analysis and/or working with machine learning-based models. This section will mainly bring forth important things to note about the data.

5.3.1 Latitude and longitude approximations

As mentioned earlier in the implementation chapter, the latitude and longitude coordinates were sampled every 20 seconds. Since the coordinates data was not sampled continuously as the event data was, an approximation of where each event took place was made. Some sort of approximation had to be made to have the possibility to add external data to each event, but how the approximation should have been done is unclear. We felt that just mapping each event to the nearest latitude and longitude sample in time was the most reasonable in terms of time and performance.

Since we map each event to the nearest latitude and longitude sample in time, this means that our approximation can give an error margin of max 10 seconds. The margin of error for each event then depends on how fast the driver is driving. An example of this is if the approximation is off by 10 seconds and the driver is driving at a speed of 100/kmh, then our estimate of the driver's location can be off by almost 280m. So for areas with higher speed limit (or for faster drivers in general), our data will be a bit noisier.

5.3.2 Biased data

The way the data is collected introduces a possibility of having biased data. The data is only collected from drivers that are willingly and knowingly being subjected to being monitored, this means that they are aware that they are monitored during their trips. There is therefore a possibility that some drivers would not drive as they normally would, instead they might try to drive as safe and legal as possible. This is of course a good thing in terms of traffic safety and such but from a data standpoint, the data would be somewhat biased or non representative of the population.

5.3.3 Noisy data

One of the most important things to keep in mind is that the data is noisy. It has been confirmed that the data for each user includes not only driver data, but passenger data as well. This means that the data received for a specific user could mean that he either drove the car, or was just a passenger during that trip. This is problematic since this means that data associated with a user could potentially be non-representative of this user's driving behavior. We have no idea how many passenger trips exists in our dataset but it is something to take into consideration when for example evaluating results and such. There are different algorithms and methods that can potentially detect anomalies in the data, so one possibility would be to do anomaly detection for each user and try to filter out trips where he/she most likely was a passenger in. But we felt that this was out of our scope for this thesis project.

5.4 Additional data

5.4.1 Open Street Map Data

As with all data, there can be incorrect values and in particularly with data that is sourced from contributions made by individual users like in the case of Open Street Map. This was considered when using the data by for instance applying different filters on max speed etc.

Another concern with the data was its geographical availability, again because the data is based on individual contributions some areas of the world will have much more sparse availability of data. This led to inaccurate data when doing GPS lookups in the kD-tree, as the search returned the nearest neighbor of the queried coordinate.

In some very large cities another problem was that there were too many roads, this coupled with the noise latitude longitude problem described in section 5.3.1 made it hard to return the correct max speed or road type in some cases. In some extreme cases there could be multiple levels of streets running on top of each other, which made it impossible to tell which one of the road the driver had actually been driving on, just using the GPS information.

6
Conclusion

This thesis work's main purpose was to investigate and develop machine learning-based methods for deriving driver behavior, based on naturalistic smartphone data. The research done throughout the thesis work resulted in two different methods, *Latent subspace clustering* and *clustering on driver norms*.

Latent subspace clustering:
This is a deep learning-based method where the main idea is to map each driver onto a latent subspace where similar drivers are closer to each other in the subspace, thus clustering on this subspace would group together similar drivers.

Clustering on driver norms:
This method focuses on modelling driver norms by analyzing how drivers drive on average in certain scenarios. By having a representation of each driver's usual driving pattern, it is possible to analyze and derive certain typical patterns for a group of drivers.

The research done also comprised the effects of external data on our methods for driver behavior profiling. The findings of the thesis work might therefore be of interest to those who also want to investigate the possible impact of combining heterogeneous sources of information to describe behavioral patterns for drivers.

The generalisability of these results is subject to certain limitations. For instance, our methods only take into account driving traits that can be derived from a driver monitoring system. Thus, a driver behavior profile, in our study, is limited to acceleration, turning and braking of a vehicle.

6.1 Suggestions for further research

Further research suggestions would be to add more contextual information to be able to more precisely analyze and understand the different behavior profiles. However, this would also introduce more contextual information in the behavior profiles. Therefore an additional suggestion to prevent this would be to only analyze drivers under the same contextual conditions. As far as the methods are concerned, for the deep learning method, *Latent subspace clustering*, one could further fine tune hyperparameters or try out other network architectures to improve the learned latent subspace. For the method *Clustering on driver norms*, one could try to engineer

more informative features since a better representation of a driver's characteristics would yield a better representation of different driver profiles.

References

[1] "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2018-04-24.

[2] N. N. C. for Statistics and Analysis, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," 2015. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

[3] W. Dong, J. Li, R. Yao, C. Li, T. Yuan, and L. Wang, "Characterizing driving styles with deep learning," *arXiv preprint arXiv:1607.03611*, 2016.

[4] G. Castignani, T. Derrmann, R. Frank, and T. Engel, "Validation study of risky event classification using driving pattern factors," *2015 IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, pp. 1–6, 2015.

[5] J. W. J. Lipkowitz and V. Sokolov, "Clusters of driving behavior from observational smartphone data," *CoRR*, vol. abs/1710.04502, 2017. [Online]. Available: http://arxiv.org/abs/1710.04502

[6] Z. Li and C. Cai, "Unsupervised detection of drivers' behavior patterns," in *Australasian Transport Research Forum (ATRF), 37th*, 2015.

[7] J. F. Júnior, E. Carvalho, B. V. Ferreira, C. de Souza, Y. Suhara, A. Pentland, and G. Pessin, "Driver behavior profiling: An investigation with different smartphone sensors and machine learning," *PLoS one*, vol. 12, no. 4, p. e0174959, 2017.

[8] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[9] A. Kattan, R. Abdullah, and Z. W. Geem, *Artificial neural network training and software implementation techniques*. Nova Science Publishers, Inc., 2011.

[10] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[11] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2018.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[14] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[16] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(98)00116-6

[17] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021068

[18] G. Hinton, "Lecture 6a overview of mini-batch gradient descent," http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, accessed: 2018-04-24.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[20] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, no. 2, pp. 129–137, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1982.1056489

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[22] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[23] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion?" *Journal of classification*, vol. 31, no. 3, pp. 274–295, 2014.

[24] C. Ding and X. He, "Cluster merging and splitting in hierarchical clustering algorithms," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on.* IEEE, 2002, pp. 139–146.

[25] D. Wishart, "256. note: An algorithm for hierarchical classifications," *Biometrics*, pp. 165–170, 1969.

[26] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[27] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[28] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[29] T. M. Kodinariya and P. R. Makwana, "Review on determining number of cluster in k-means clustering," *International Journal*, vol. 1, no. 6, pp. 90–95, 2013.

[30] "Open street map," https://www.openstreetmap.org, accessed: 2018-04-24.

[31] "Overpass ql," https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL, accessed: 2018-04-24.

[32] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 4368–4374.

[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[34] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

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

A

Appendix 1

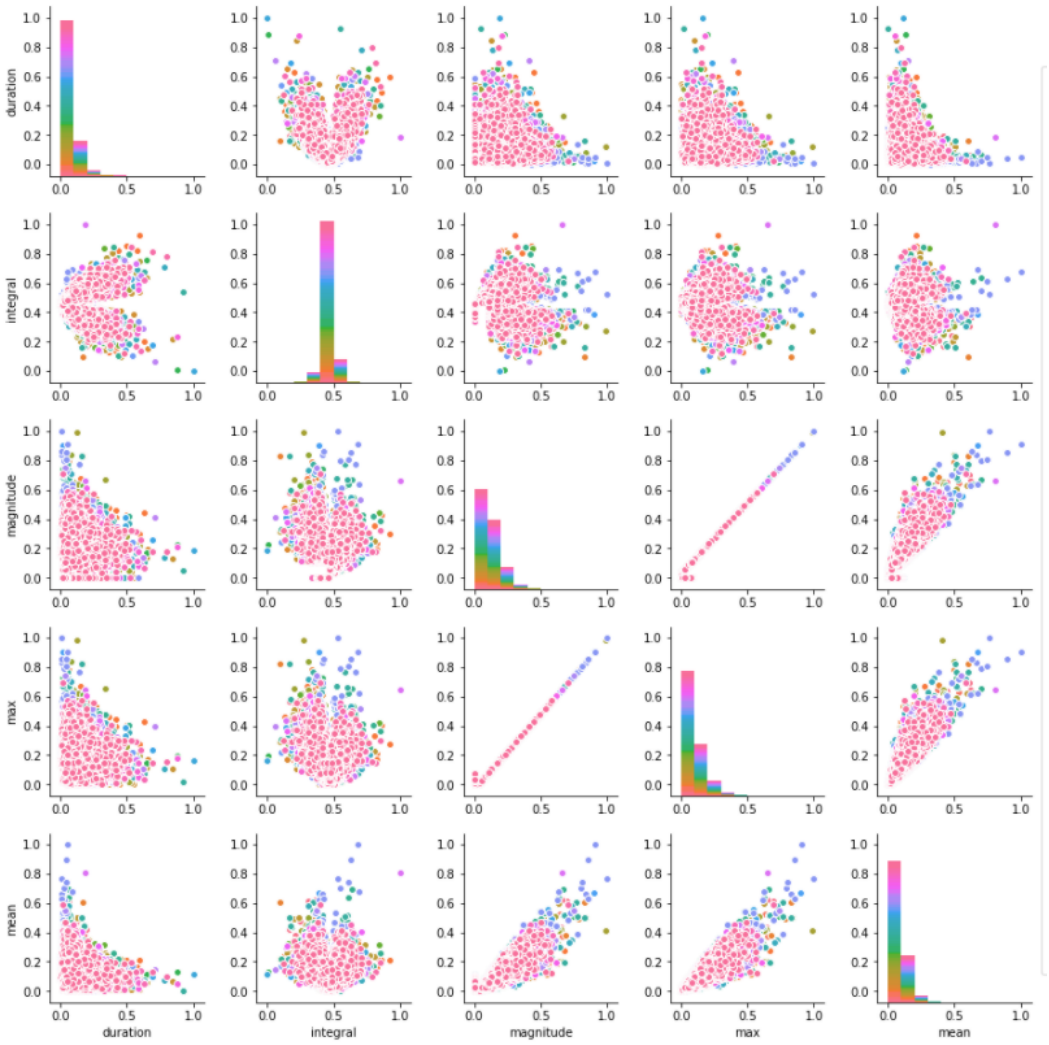


Figure A.1: An image showing the correlation between all features for 100 users.

B

Appendix 2

B.1 Acceleration histograms

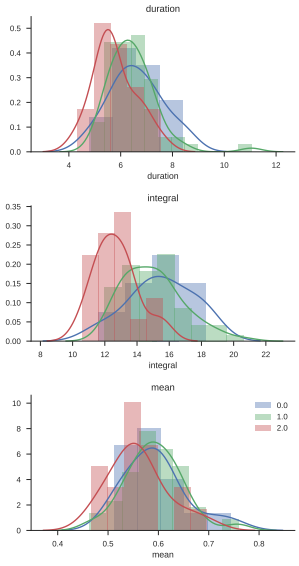


Figure B.1: Net 3

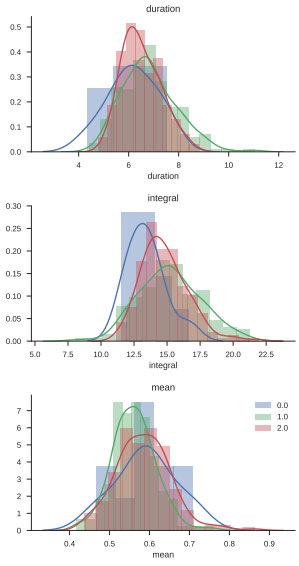


Figure B.2: Net 5

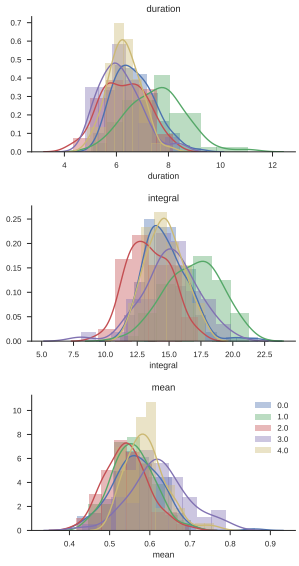


Figure B.3: Net 7

Figure B.4: The histograms in each column displays the feature distributions of the different clusters for their acceleration events. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.

B.2 Turn histograms

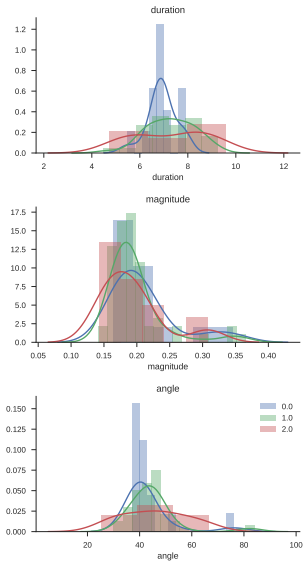


Figure B.5: Net 3

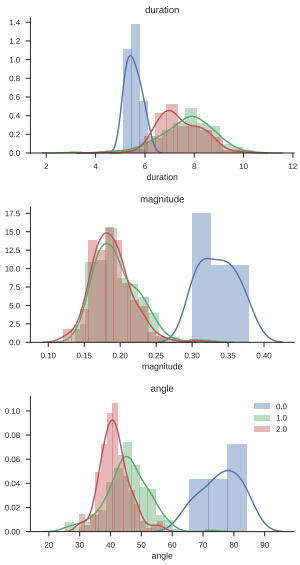


Figure B.6: Net 5

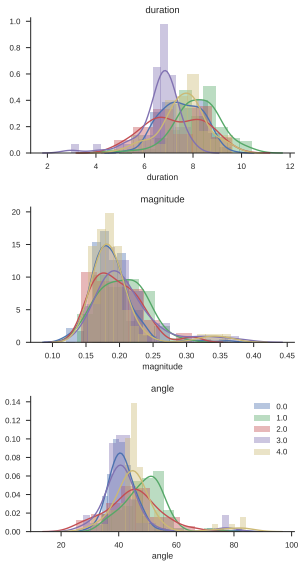


Figure B.7: Net 7

Figure B.8: The three histograms in each column display the feature distributions of the different clusters for their turn events. The first histogram from the top shows the duration, the second histogram shows the magnitude and the last histogram shows the angle.

C

Appendix Net 3

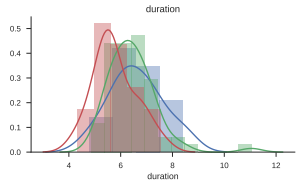


Figure C.1: *Net 3: All acceleration events*

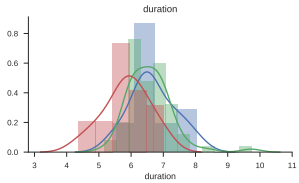


Figure C.2: *Net 3: All brake events*

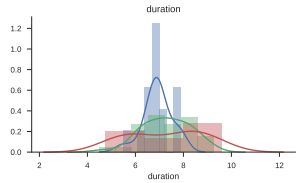


Figure C.3: *Net 3: All turn events*

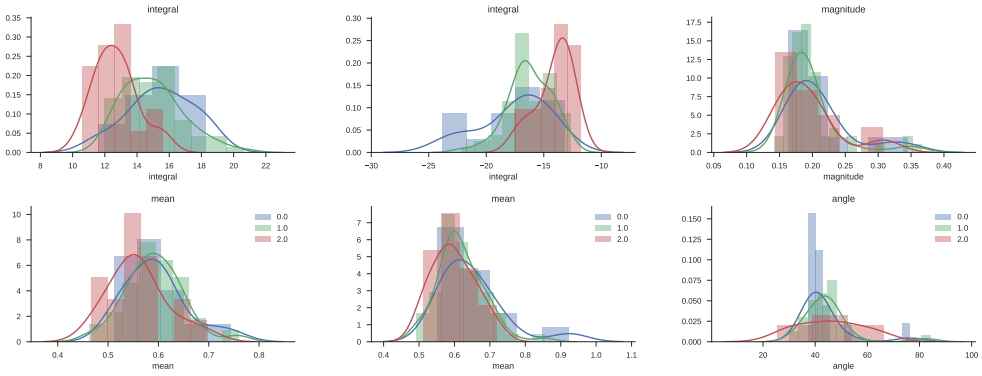


Figure C.4: *The histograms in the first two column displays the feature distributions of all the acceleration and break events. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

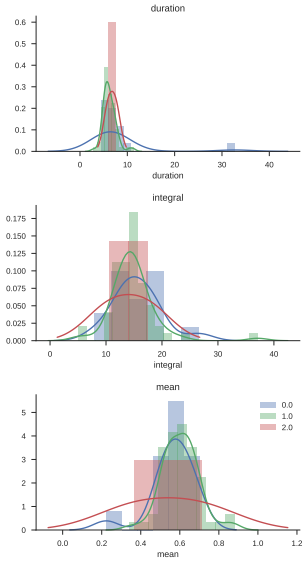


Figure C.5: *Net 3: 0-30km/h*

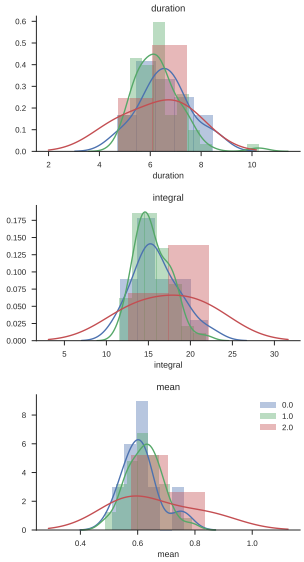


Figure C.6: *Net 3: 30-70km/h*

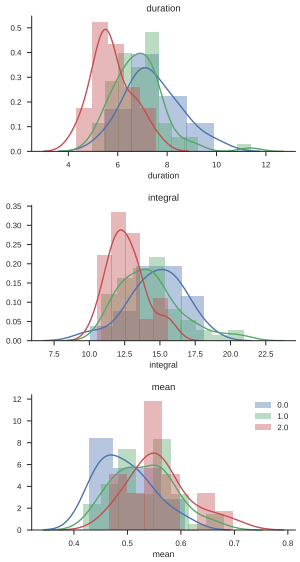


Figure C.7: *Net 3: 70-180km/h*

Figure C.8: *The histograms in each column displays the feature distributions of the acceleration events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

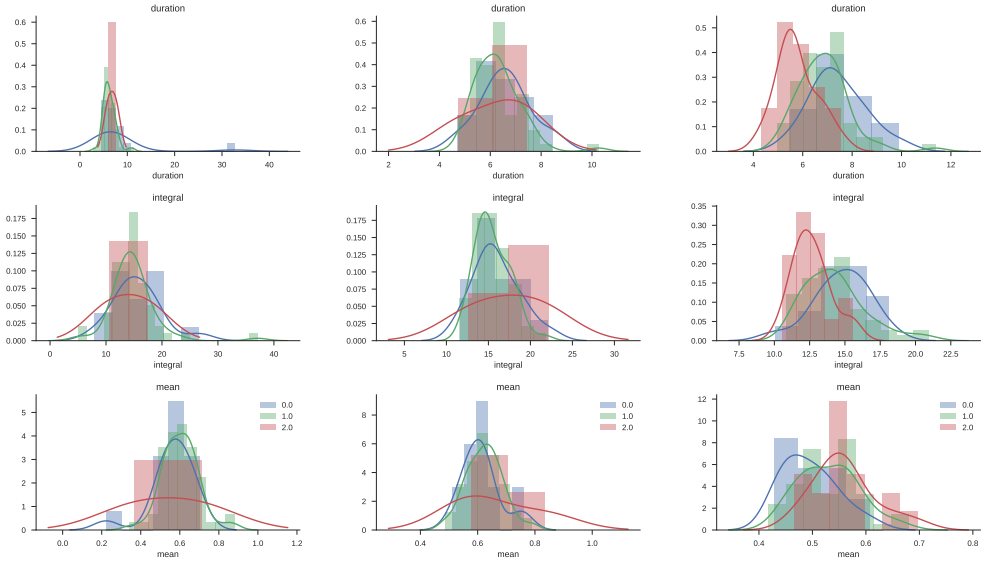


Figure C.9: *Net 3: 0-30km/h*

Figure C.10: *Net 3: 30-70km/h*

Figure C.11: *Net 3: 70-180km/h*

Figure C.12: *The histograms in each column displays the feature distributions of the break events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

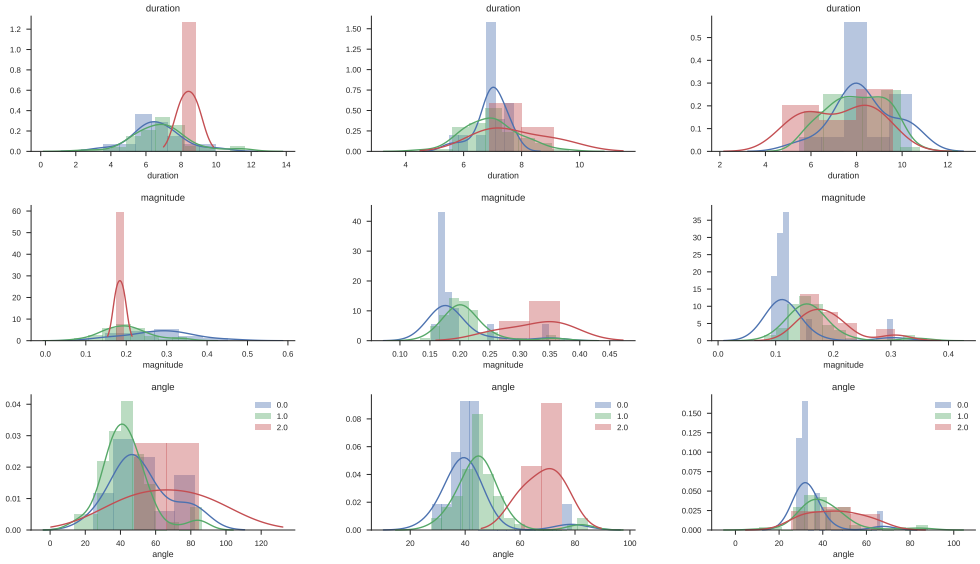


Figure C.13: *Net 3: 0-30km/h*

Figure C.14: *Net 3: 30-70km/h*

Figure C.15: *Net 3: 70-180km/h*

Figure C.16: *The histograms in each column displays the feature distributions of the turn events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the magnitude and the third histogram the angle.*

D

Appendix Net 5

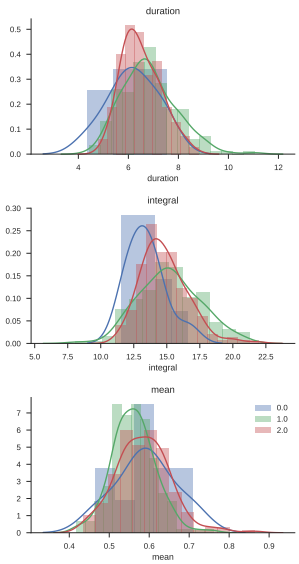


Figure D.1: *net 5: All acceleration events*

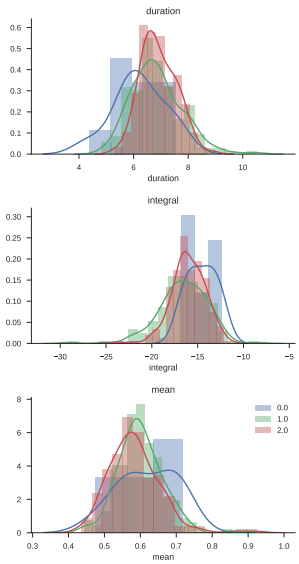


Figure D.2: *net 5: All brake events*

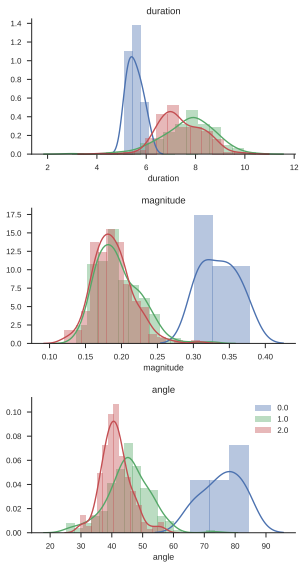


Figure D.3: *net 5: All turn events*

Figure D.4: *The histograms in the first two columns display the feature distributions of all acceleration and break events. The histograms in the last column show the distribution of all the turn events. For the two first columns the first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean. For the last column the first histogram show the duration feature, the second histogram the magnitude and the last histogram the angle.*

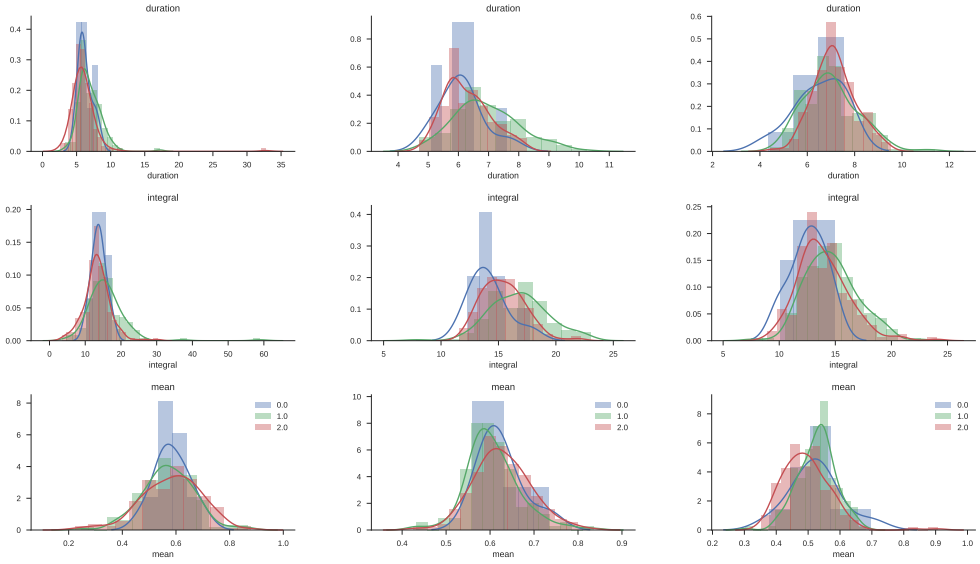


Figure D.5: *net 5: 0-30km/h*

Figure D.6: *net 5: 30-70km/h*

Figure D.7: *net 5: 70-180km/h*

Figure D.8: *The histograms in each column display the feature distributions of the acceleration events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

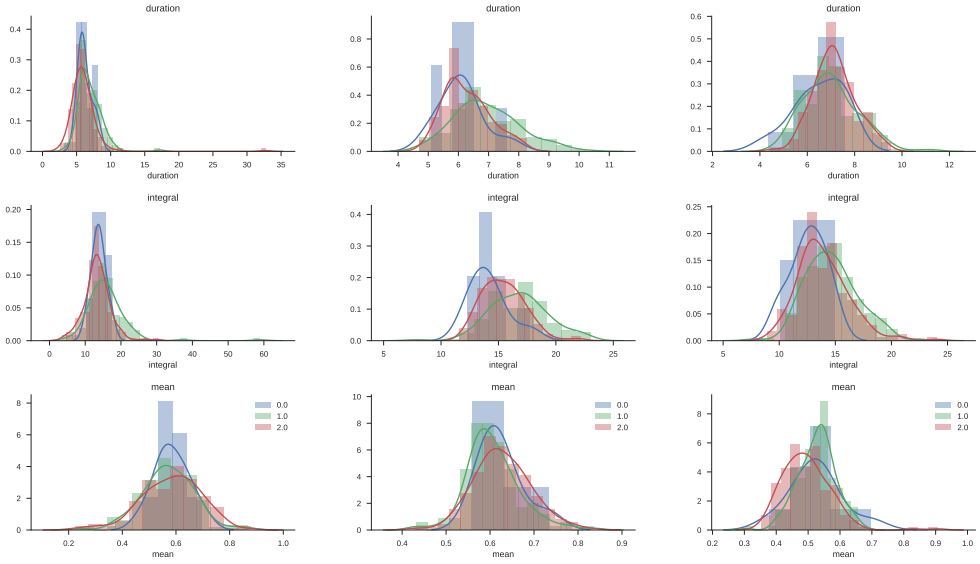


Figure D.9: *net 5: 0-30km/h*

Figure D.10: *net 5: 30-70km/h*

Figure D.11: *net 5: 70-180km/h*

Figure D.12: *The histograms in each column display the feature distributions of the break events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

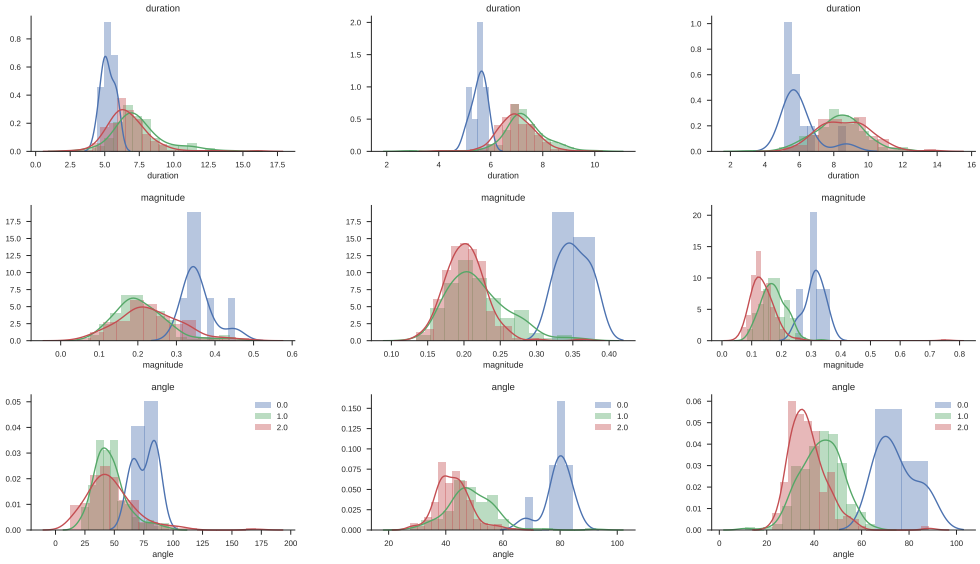


Figure D.13: *net 5: 0-30km/h*

Figure D.14: *net 5: 30-70km/h*

Figure D.15: *net 5: 70-180km/h*

Figure D.16: *The histograms in each column display the feature distributions of the turn events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the magnitude and the third histogram the angle.*

E

Appendix Net 7

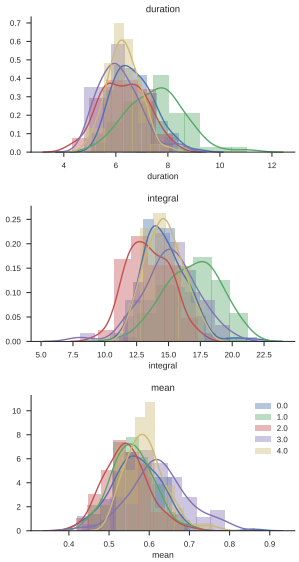


Figure E.1: *net 7: All acceleration events*

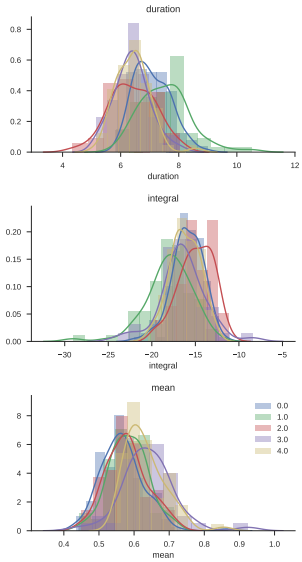


Figure E.2: *net 7: All brake events*

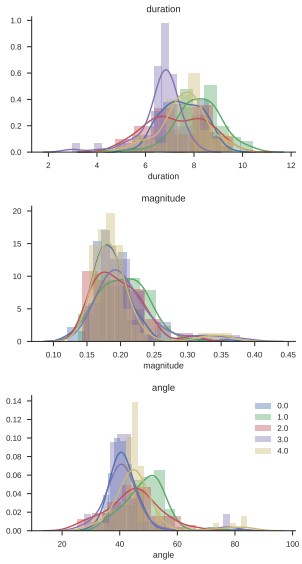


Figure E.3: *net 7: All turn events*

Figure E.4: *The histograms in the first two columns display the feature distributions of all acceleration and break events. The histograms in the last column show the distribution of all the turn events. For the two first columns the first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean. For the last column the first histogram show the duration feature, the second histogram the magnitude and the last histogram the angle.*

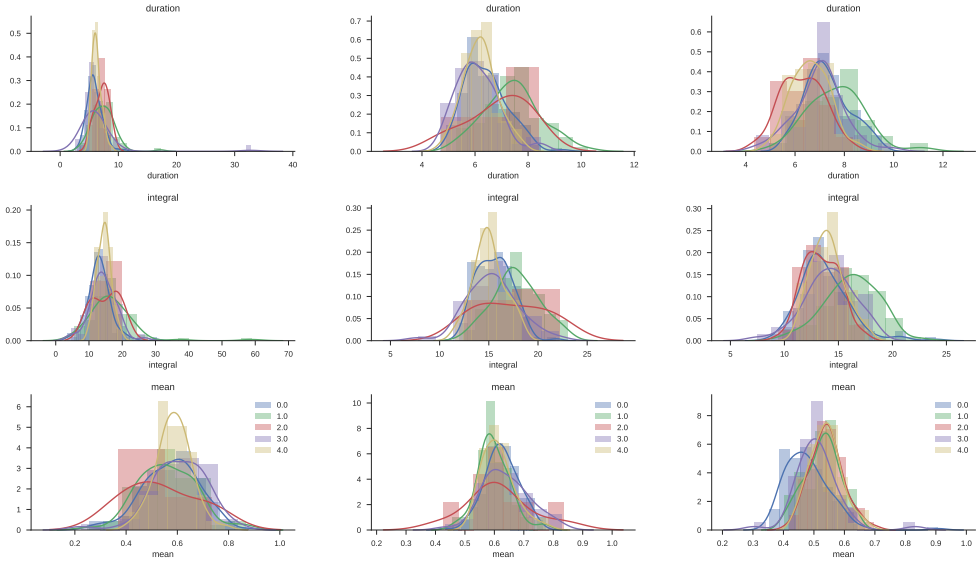


Figure E.5: *net 7: 0-30km/h*

Figure E.6: *net 7: 30-70km/h*

Figure E.7: *net 7: 70-180km/h*

Figure E.8: *The histograms in each column display the feature distributions of the acceleration events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

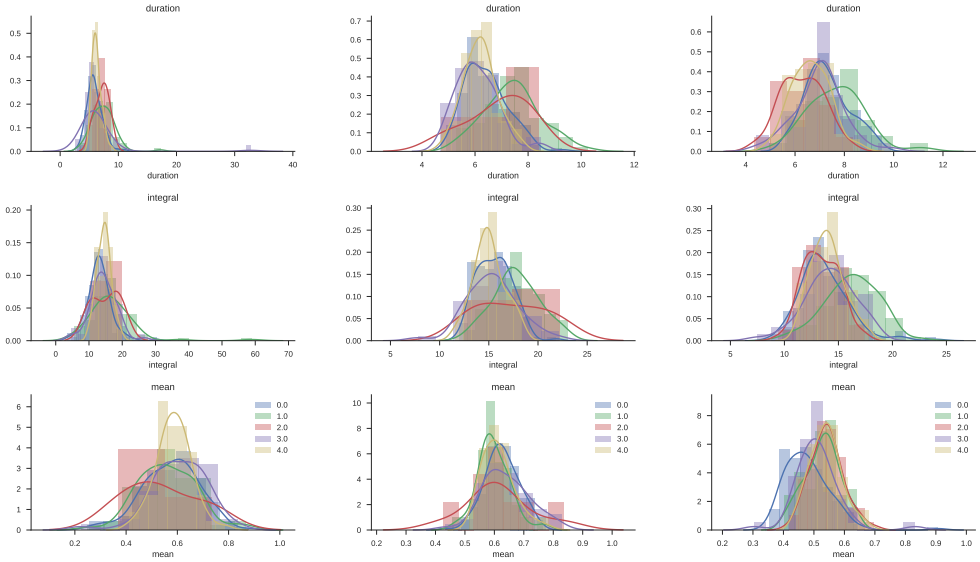


Figure E.9: *net 7: 0-30km/h*

Figure E.10: *net 7: 30-70km/h*

Figure E.11: *net 7: 70-180km/h*

Figure E.12: *The histograms in each column display the feature distributions of the break events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the integral and the third histogram the mean.*

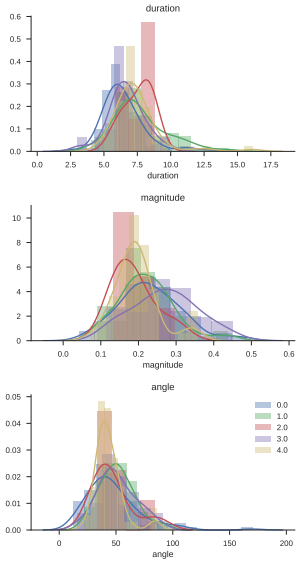


Figure E.13: *net 7: 0-30km/h*

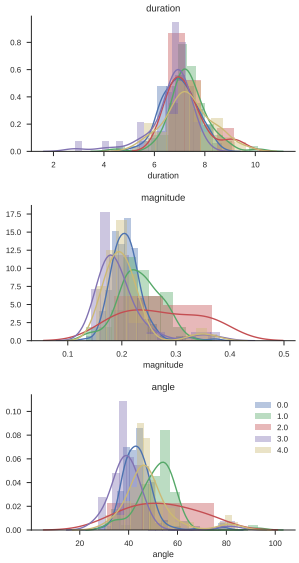


Figure E.14: *net 7: 30-70km/h*

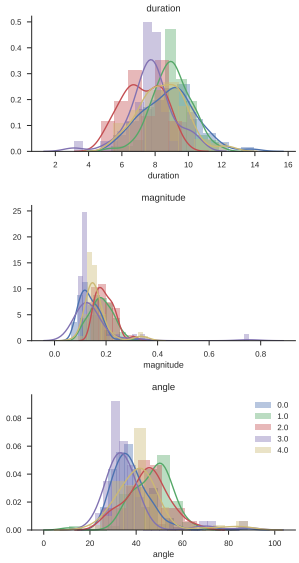


Figure E.15: *net 7: 70-180km/h*

Figure E.16: *The histograms in each column display the feature distributions of the turn events filtered on maximum speed allowed. The first histogram from the top display the duration feature, the second histogram the magnitude and the third histogram the angle.*

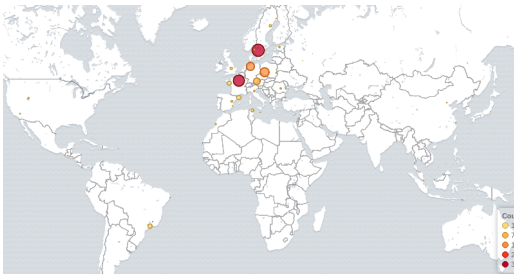


Figure E.17: *A figure showing cluster 0's driver's geographical location for net 7 using Kibana.*

Figure E.18: *A figure showing cluster 1's driver's geographical location for net 7 using Kibana.*

Figure E.19: *A figure showing cluster 2's driver's geographical location for net 7 using Kibana.*

Figure E.20: *A figure showing cluster 3's driver's geographical location for net 7 using Kibana.*

Figure E.21: *A figure showing cluster 4's driver's geographical location for net 7 using Kibana.*