Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2017

Lane Change Intent Analysis for Preceding Vehicles

- a Study Using Various Machine Learning Techniques

Fredrik Ljungberg

Master of Science Thesis in Electrical Engineering

Lane Change Intent Analysis for Preceding Vehicles - a Study Using Various Machine Learning Techniques

Fredrik Ljungberg

LiTH-ISY-EX--17/5059--SE

Supervisor: **Martin Lindfors**
ISY, Linköpings universitet
Joseph Ah-King
Scania CV AB
Christian Larsson
Scania CV AB

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In recent years, the level of technology in heavy duty vehicles has increased significantly. Progress has been made towards autonomous driving, with increased driver comfort and safety, partly by use of advanced driver assistance systems (ADAS).

In this thesis the possibilities to detect and predict lane changes for the preceding vehicle are studied. This important information will help to improve the decision-making for safety systems. Some suitable approaches to solving the problem are presented, along with an evaluation of their related accuracies.

The modelling of human perceptions and actions is a challenging task. Several thousand kilometers of driving data was available, and a reasonable course of action was to let the system learn from this off-line. For the thesis it was therefore decided to review the possibility to utilize a branch within the area of artificial intelligence, called supervised learning. The study of driving intentions was formulated as a binary classification problem. To distinguish between lane-change and lane-keep actions, four machine learning-techniques were evaluated, namely naive Bayes, artificial neural networks, support vector machines and Gaussian processes. As input to the classifiers, fused sensor signals from today commercially accessible systems in Scania vehicles were used.

The project was carried out within the boundaries of a Master's Thesis project in collaboration between Linköping University and Scania CV AB. Scania CV AB is a leading manufacturer of heavy trucks, buses and coaches, alongside industrial and marine engines.

Acknowledgments

I would like to show my deepest gratitude to my supervisors, both at Scania CV AB and at Linköping University. Joseph Ah-King and Christian Larsson, at Scania CV AB, for all their input during the work with this thesis and Martin Lindfors, at Linköping University, for his guidance and feedback during the thesis work. It has been invaluable. I would also like to thank my examiner, Daniel Axehill, for his shown interest.

At last, but not least, I would like to thank my fellow thesis writers at Scania CV AB for making the thesis period a very enjoyable experience. Among these a special thank you is directed to Klas Lindsten (Linköping University), Gustav Ling (Linköping University) and Ermin Kodzaga (Uppsala University) for, without exceptions, always being ready to bounce ideas about our thesis projects, life, the universe and everything.

Linköping, June 2017
Fredrik Ljungberg

Contents

Notation

Abbreviations

Abbreviation	Meaning
ADAS	Advanced driver assistance systems
AI	Artificial intelligence
CAN	Controller area network
GP	Gaussian process
GPS	Global positioning system
HMM	Hidden Markov model
IMU	Inertial measurement unit
MAP	Maximum a posteriori
ML	Maximum likelihood
NB	Naive Bayes
RADAR	Radio detection and ranging
ROC	Receiver operating characteristics
RVM	Relevance vector machine
SVM	Support vector machine

1

Introduction

During recent years the production industry of heavy duty vehicles has taken big steps in development. Progress has been made towards autonomous driving, with increased driver comfort and safety, partly by use of advanced driver assistance systems (ADAS). Artificial intelligence (AI) such as machine learning is finding its way into many modern decision and control problems and upcoming ADAS will increasingly drive and steer the vehicle to help and relieve the driver.

One possible future solution for heavy duty vehicles is that they will have an ADAS function implemented that helps the driver with lane-following on highways. Laterally, the truck will follow the lane that it is currently in, as long as the lane-markers are visible to the sensors. If there are no clear lane-markers available, the preceding vehicle will be followed in the meantime. Issues do arise if the preceding vehicle, during that time, leaves the current lane because the system, in its current state, cannot acknowledge this and hence cannot act accordingly. Ways to predict the intention of the vehicle in front of oneself is therefore sought in order to either trigger a new system-function or at least to alert the driver of a potential upcoming highway drop-off.

A heavy duty vehicle produced by Scania CV AB according today's standards is equipped with several sensors such as GPS/IMU, camera, RADAR etc. and well performing ways to track the motion of nearby objects are already developed. By observing a preceding vehicle, using these sensors, it might be possible to determine the intentions of its driver. This thesis aims at investigating this possibility further. The project was carried out within the boundaries of a Master's Thesis project in collaboration between Linköping University and Scania CV AB.

1.1 Background

Steering manoeuvres can be seen as an implementation of the driver's intention. The intention cannot be observed directly, since it is an inner state of the driver and has to be inferred from observable signals from the environment in which the vehicle operates [7]. It is not sufficient to rely exclusively on the turn signal for recognition of intentions, because even if it is legally mandatory to use in most countries, many drivers tend to not to use the signal with any consistency. The turn indicator is actually only used in two thirds of all lane changes according to Olsen [22]. In addition it is possible that the situations that are most dangerous and hence most interesting to catch are the ones in which the manoeuvre is not appropriately announced.

A deep-rooted branch within the area of artificial intelligence is pattern recognition. It is used for inductive inference where predictions are based on observations. The field of science is concerned with the automatic discovery of correlations in data through the use of computer algorithms. It also includes the usage of these regularities in order to take actions [3].

Several thousand kilometers of driving data was available. Thus, a reasonable course of action was to let the system learn from this off-line. There exist a multitude of approaches for successfully addressing supervised learning in a variety of contexts. Some of the most popular are: decision trees [21], neural networks [15], support vector machines [3] and nearest neighbour [13]. There are also methods that do not require the assumption of independent and identically distributed data points, for example hidden Markov models [3].

1.2 Related Work

Intelligent vehicle systems have been a topic of research for some time and envelope a wide area of research topics. The latest sensors, computer technologies and artificial intelligence algorithms, instrumented and implemented on autonomous vehicles, in a set of emulated urban driving scenarios are discussed by Anhal et. al., [1] while Meng et. al., [17] explores the technical feasibility of five advanced driver assistance system functions to contribute to road traffic safety.

The lateral and longitudinal dynamics and control of ground vehicles have been studied thoroughly over the past decades by, for example, Fenton. [11] [12]

For the development of road vehicles and the study of their dynamic behaviour, it is necessary to understand the interaction between driver and vehicle dynamics. When human drivers act they take a lot of information provided by the environment into account. The driver actions are also based on anticipation and adapted to the dynamics of the particular vehicle. The modelling of human perceptions and actions are challenging tasks. [18] and [8] attempt to model steering behaviour and find parameters to characterize the behaviour of a typical driver.

Other current approaches are more data-driven and not model-based. [28] analyses intents to change lane by using a form of Bayesian learning and [7] [23] [4] propose the usage of hidden Markov models (HMMs). [28] suggests the usage of cameras to analyse the driver's head motion but since this thesis is about predicting the intentions of other road users and not concerning the own vehicle, this information is inaccessible.

Attempts have also been made using artificial neural networks (ANNs). An ANN was in [9] used for drowsiness detection learned by driver steering, in [24] to design the longitudinal and lateral controller for an autonomous vehicle and in [6] to develop a trajectory set of human-like lane changes, learned from driving data of different drivers.

This work aimed to give a contribution in this field by delimiting the observable signals to longitudinal and lateral movement of the preceding vehicle, at times along with road surface markings and GPS data. Also, even though previous works on drivers' intention recognition have shown promising potential by using various methods of pattern recognition, most of them have focused on the own vehicle and in that respect this thesis can complement earlier investigations.

1.3 Delimitations

For this thesis, only lane-switches that occur on highways including entrance and exit ramps were dealt with. Tracked vehicles were also assumed to follow European regulations and laws, excluding the usage of turn signals. Only sensor data from today, commercially accessible systems in Scania vehicles was used for testing. Since some of the sensors required for tracking might lose performance in poor light conditions or certain weather types, like fog and snow, the survey was also constrained to only include cases when the tracking is persistently substantial, i.e. cases when estimates of the preceding vehicle's position and motion are available with moderate frequency. Since the learning will take place off-line, capability evaluation of the selected approaches was mainly focused on classification accuracy and not on processing time.

1.4 Approach

The work to be carried out within the scope of this thesis was to implement and evaluate a suitable algorithm, for prediction of lane change intentions of the preceding vehicle. The workflow included literature study, concept evaluation, concept selection, data processing, testing and analysis. For development and evaluation, multiple thousand kilometers of logged driving data from different Scania trucks was available. A test system was built using Matlab.

First and foremost a more thorough study of related research was carried out in order to determine the different approaches which are available to solve this problem. A vast number of approaches exist in books about machine learning [15] [13] [3] [21]. By studying reports about work in adjacent areas, [28] [7] [23] [4], it seemed like just a handful of methods were actually successful in practice for driver intention analyses. This might be because they were the only ones that were suitable or it might be a coincidence. The hypothesis was that some approach that had yet to be tested might prove to be the most beneficial. The idea was to identify a couple of promising techniques and to compare their usefulness.

2

Classification Preliminaries

The following chapter will summarise the basic theory needed to understand the methods used in this thesis. First an introduction to machine learning and pattern recognition is provided. This is followed by brief descriptions of a selection of approaches. After that some metrics for result comparisons are given. Lastly a short introduction to classification using multiple classes is presented.

Notation	Description
x_i^j	The ith training input of the jth covariate
y_i	The ith output
$\mathbf{x}$	The full covariate vector
Z	A stochastic version of z
C_c	Classification label c
f	True function
h	Hypothesis function
N	Number of example input-output pairs
N_c	Number of classes
N_{cov}	Number of covariates
θ	Vector of unknown parameters to learn
$\mathbb{D}$	Training data set

Table 2.1: *Symbols and notations used for pattern recognition.*

2.1 Introduction to Pattern Recognition

According to [21] there are three types of feedback that determine the three main types of learning. Clustering is the most common unsupervised learning task and is a way to detect potentially useful clusters of input examples. In unsupervised learning a pattern in the input is learned despite the fact that no feedback is supplied. Another way of learning is by using a series of punishments and rewards. It is then required to specify a reward function. This is called reinforcement learning. The third way of learning is supervised. By presenting to the system some example input-output pairs it can learn a function with a general mapping. This was supposed to be the most suitable way of training the sought model.

More formally, assume that a training set of N example input-output pairs

$$\mathbb{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\},$$

is available. Also assume that each y_i was generated by an unknown function,

$$y = f(\mathbf{x}, e).$$

Here e is noise that originates from the fact that the intention is inferred, from observable signals and no such observation will indicate a lane switch with complete certainty. The goal with supervised learning is to discover a hypothesis function, $h(\mathbf{x})$, that approximates the true function $f(\mathbf{x}, e)$. When the output, y, is continuous the problem is called regression and when it is a set of discrete, finite values the problem is called classification. In order to be able to predict the behaviour of a preceding vehicle a distinct difference must be found between how drivers operate their vehicles while assessing and preparing for a steering maneuver and how they operate their vehicles when they are not. The analysis of

driver intentions can therefore be formulated as a binary classification problem. An illustration of a linear decision rule between two separated sets of classes are provided in Figure 2.1

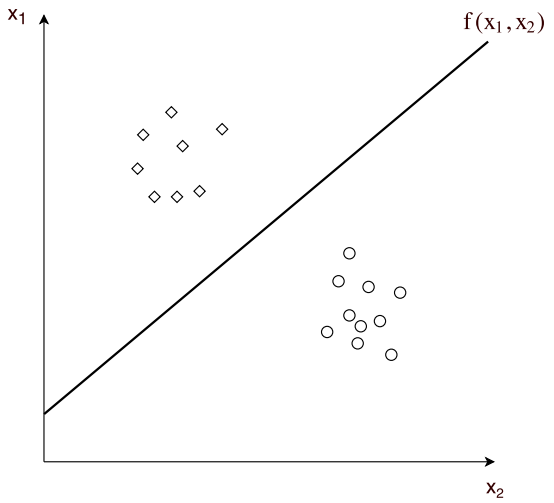

Figure 2.1: *Visualization of a typical linear decision rule for binary classification. Diamonds make up one class and circles make up another.*

Classification asserts that similar input, called covariate vectors, belong in the same class, C, while other dissimilar covariate vectors are contained in others. Each $\mathbf{x}_i$ in the training sample is associated with an error given a hypothesis function. This is usually characterized by a cost function $V(y, h(\mathbf{x}, \theta))$ like, for example, the squared Euclidean distance, $V(y, h(\mathbf{x}, \theta)) = |y - h(\mathbf{x}, \theta)|^2$. Training the model means finding the hypothesis function, with belonging parameter values, that minimizes the expected value of the error at each x,

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E_{Y|X}\{V(y, h(\mathbf{x}, \theta)\} \tag{2.1}$$

2.1.1 Probabilistic Learning

In probabilistic learning it is supposed that the covariate vector and the corresponding labels, $(\mathbf{X}, Y)$, are stochastic variables represented by some joint probability density $\Pr(\mathbf{X}, Y)$. This seems feasible because the input, $\mathbf{x}$, consists of fused sensor readings which includes errors in measurements. In fact, even if $\mathbf{x}$ was assumed to be assured the driver's intention, y, would not be certain. The fact that the drivers intention is a state that is not fully observable, justifies probabilistic learning. This is because a probabilistic method might be able to model the uncertainty, e, as well. The joint probability density is given by

$$\Pr(\mathbf{X}, Y) = \Pr(Y|\mathbf{X}) \Pr(\mathbf{X}), \tag{2.2}$$

and this way supervised learning can be formally characterized as a density estimation problem where one is concerned with determining the properties of the conditional density $\Pr(Y|\mathbf{X})$.

For a distribution, $\Pr(\mathbb{D}|\theta)$, parametrised by θ, and training data, $\mathbb{D} = \{\mathbf{x}_{train}, \mathbf{y}_{train}\}$, learning in probabilistic classification corresponds to inferring the θ that best explains the data $\mathbb{D}$. There are various criteria for defining this but for this thesis the two most common decision rules will be sufficient.

- *Maximum A Posteriori (*MAP*)*: This is a summation of the posterior, that is

$$\theta^{MAP} = \underset{\theta}{\operatorname{argmax}} \Pr(\theta|\mathbb{D})$$

- *Maximum Likelihood (*ML*)*: Assuming a flat, constant, prior, $P(\theta) = c_0$, the MAP solution is equivalent to setting θ to the value that maximises the likelihood of observing the data.

$$\theta^{ML} = \underset{\theta}{\operatorname{argmax}} \Pr(\mathbb{D}|\theta)$$

Bayesian Learning

Rather than choosing the most likely model or delineating the set of all models that are consistent with the training data, one approach is to compute the posterior probability of each model given the training examples. In contrast to aforementioned approaches Bayesian learning simply calculates the probability of each hypothesis, given the data, and makes predictions on that basis. In other words the predictions are made by using all the hypothesis functions, weighted by their probabilities, rather than solely using the best one. The hypotheses themselves are essentially intermediaries between the raw data and the predictions [21].

2.1.2 Selection of method

Using Bayes' theorem (2.2) can be formulated either as $\Pr(Y|\mathbf{X}) \Pr(\mathbf{X})$ or as $\Pr(\mathbf{X}|Y)Pr(Y)$ which gives rise to two different approaches. The first approach is called the discriminative approach and focuses on modelling $\Pr(Y|\mathbf{X})$ directly. The second one, which is known as the generative approach, models the class-conditional distributions, $\Pr(\mathbf{X}|Y)$, together with the *prior* probabilities of each class, $\Pr(Y)$. The *posterior* probability for each class can then be inferred as

$$\Pr(Y|\mathbf{X}) = \frac{\Pr(\mathbf{X}|Y) \Pr(Y)}{\Pr(\mathbf{X})} = \frac{\Pr(\mathbf{X}|Y) \Pr(Y)}{\sum_{c=1}^{N_c} \Pr(\mathbf{X}|C_c) \Pr(C_c)} \tag{2.3}$$

Because $\Pr(\mathbf{X}|Y) \Pr(Y) = \Pr(\mathbf{X}, Y)$ this is equal to explicitly modeling the actual distribution of each class.

Both these methods are correct but it is possible to identify some advantages and drawbacks with the two. Something appealing about the discriminative approach is that it directly models the sought after density, $\Pr(Y|\mathbf{X})$. Despite that, in order to deal with unlabelled data points, outliers and missing input values in a principled fashion it is useful to have $\Pr(\mathbf{X})$ available which can be obtained from marginalizing out the class label Y from the joint density, since $\Pr(\mathbf{X}) = \sum_y \Pr(Y)\Pr(\mathbf{X}|Y)$, in the generative approach. An issue with the generative approach is that density estimation for the class-conditional probability distributions is a difficult problem. This is especially significant when $\mathbf{X}$ is of high dimension. When classification is the sole interest this means that the generative approach may require solving a problem that is harder than necessary. An important factor when it comes to deciding upon an approach is also the conductivity to incorporation of any, possibly available, prior information [27].

To turn any of these approaches into practical methods models are required, either for the conditional probability $\Pr(Y|\mathbf{X})$ or for the distribution $\Pr(\mathbf{X}, Y)$ and these can either be of parametric or non-parametric form. Parametric models assume some finite set of parameters, θ, and given the parameters, future predictions are independent of the observed data. This means that the complexity of the model is bounded even if the amount of data is unbounded. This lack of flexibility leads to an important limitation, which is that the chosen density can be a poor model of the distribution that generates the data, which in turn can result in bad predictive performance [3]. When data sets are small however, it makes sense to have a strong restriction on the allowable hypotheses in order to avoid overfitting [21]. Overfitting is a problem that occurs when the solution is too customized for the training data and not generalized for, previously, unseen test data.

Non-parametric models assume that the data distribution cannot be defined in terms of a finite set of parameters. They can however often be defined by assuming an infinite dimensional θ. Usually θ is thought of as a function. The amount of information that θ can capture about the data, D, can grow as the amount of data grows which increases flexibility.

2.2 Description of Approaches

Due to the limited amount of working hours a bounded set of methods were selected for further concept evaluation. The artificial neural network, ANN, is a popular parametric model that has proved to work well in related research, [9] [6]. The support vector machine, SVM, is currently the most popular commercially available packaged solution and is a great model if no specialized knowledge about the domain is available [21]. The property of being nonparametric makes it robust to overfitting.

Much of the basic theory and many algorithms are shared between the statistics and the machine learning community. The primary differences are perhaps the types of the problems attacked and the goal of learning. Bayesian models in some sense bring together work in the two communities [27]. For intention analyses it is convenient to seek both an estimate of y and its related level of certainty. The Bayesian framework is, mathematically, closely related to many well known machine learning models, including the ANN and the SVM but its usage is not yet very widespread. The naive Bayes method serves as a common introductory technique to the Bayesian repository. A more advanced Bayesian method is using Gaussian processes for machine learning. The usage of Gaussian processes, in statistics, can perhaps be traced back as far as the end of the 19th century but the application to real problems is still in its early phases.

Each of these techniques will be described briefly in the following sections. The general idea behind learning and prediction is presented alongside an evaluation of the advantages and disadvantages for each method respectively.

2.2.1 Naive Bayes

When the dependency relationships among the covariates used by a classifier are unknown, a possibility is taking the simplest assumption available, namely that the covariates are conditionally independent given the category. Using Bayes' theorem and the conditional independence assumption gives,

$$\Pr(Y|x^1,...,x^{N_{cov}}) = \frac{\Pr(Y)\Pr(x^1,...,x^{N_{cov}}|Y)}{\Pr(x^1,...,x^{N_{cov}})} = \frac{\Pr(Y)\prod_j \Pr(x^j|Y)}{\Pr(x^1,...,x^{N_{cov}})}. \tag{2.4}$$

This is known as the naive Bayes method, NB. Prediction is done by computing the probability for each class, using (2.4), and then simply selecting the most likely one. The denominator, $\Pr(x^1,...,x^{N_{cov}})$, is not dependent on the class and is a constant if the values of the covariates are known. The posterior probability is therefor proportional to the numerator, i.e.,

$$\Pr(Y|x^1,...,x^{N_{cov}}) \propto \Pr(Y)\prod_j \Pr(x^j|Y). \tag{2.5}$$

This information is useful because it means that the prediction can be done more compactly by ignoring denominator and instead focusing only on the numerator.

Learning and Prediction

To estimate the parameters one must assume a probability distribution or generate nonparametric models for the covariates from the training set. When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution and in the discrete case Bernoulli and multinomial distributions are popular choices [21].

Assuming a labelled training set containing a single continuous covariate, x. Let μ_c be the mean of the values in x associated with class c and σ_c^2 be the variance. Then the probability distribution of observing x given a class c can be computed using the equation for a Normal distribution parametrized by μ_c and σ_c^2.

$$\Pr(X|Y = C_c) = \frac{1}{\sqrt{2\pi}\sigma_c} e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}. \tag{2.6}$$

The likelihood of a hypothesis h is equal to the probability density assumed for the observed outcomes given that hypothesis, that is

$$L(h|X) = \Pr(X|h). \tag{2.7}$$

Let the observed values associated with class c be $x \in \{x_1, ..., x_N\}$. Given the hypothesis that the covariate, x, belongs to class c the likelihood can be written, according to (2.6) (2.7), as

$$L(Y = C_c|x) = \prod_{i=1}^{N} Pr(x_i|Y = C_c) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_c} e^{-\frac{(x_i-\mu_c)^2}{2\sigma_c^2}}. \tag{2.8}$$

In order to reduce the product to a sum, which is easier to maximize, it is convenient to look at the logarithm of the likelihood. Because the logarithm is a monotonically increasing function, the logarithm of a function achieves its maximum value at the same points as the function itself. The logarithm of the likelihood is

$$\log(L) = \sum_{i=1}^{N} \log[\frac{1}{\sqrt{2\pi}\sigma_c} e^{-\frac{(x_i-\mu_c)^2}{2\sigma_c^2}}] = N(-\log(\sqrt{2\pi}) - \log(\sigma_c) - \sum_{i=1}^{N} \frac{(x_i - \mu_c)^2}{2\sigma_c^2}. \tag{2.9}$$

Setting the derivatives to zero gives

$$\frac{d\log(L)}{d\mu_c} = -\frac{1}{\sigma_c^2} \sum_{i=1}^{N} (x_i - \mu_c) = 0 \Rightarrow \mu_c = \frac{\sum_{i=1}^{N}(x_i)}{N}, \tag{2.10}$$

$$\frac{d\log(L)}{d\sigma_c} = -\frac{N}{\sigma_c} + \frac{1}{\sigma_c^3} \sum_{i=1}^{N} (x_i - \mu_c)^2 = 0 \Rightarrow \sigma_c = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu_c)^2}{N}}, \tag{2.11}$$

i.e the maximum-likelihood value of the mean is the sample average and the standard deviation is the square root of the sample variance. [21]

As mentioned earlier, a deterministic prediction is available by first computing the probability for each class, using (2.5), and then simply selecting the most likely one. Combining the naive Bayes probability model with the maximum a posteriori decision rule like this gives a prediction

$$\hat{y} = \underset{c \in \{1,\dots,N_c\}}{\operatorname{argmax}} Pr(Y = C_c) \prod_i Pr(x_i | Y = C_c). \tag{2.12}$$

Strengths and Weaknesses

The naive Bayes method is very intuitive and easy to implement. Both training of models and prediction is fast and naive Bayes learning systems have no problems with noisy or missing data. The training is fast primarily because no search is required in order to find the maximum-likelihood naive Bayes hypothesis. Maximum-likelihood training can actually be done in linear time which is better than many other types of classifiers. The models can also provide probabilistic predictions when appropriate. Naive Bayes is a simple technique and as the name suggests the assumption of individually independent covariates is naive. Still, even if the covariates are not really independent, the algorithm has turned out to do surprisingly well in a wide range of applications [13]. Henceforth it is a convenient method to implement first and later use as a reference [21] [3].

2.2.2 Artificial Neural Networks

The usage of neural networks is an approach loosely mimicking the way a biological brain solves problems with large clusters of neurons connected by axons. This computation is done in an entirely different way than in the conventional digital computer [15].

A neural network is a collection of nodes connected together. Every node is part of a network as the one illustrated in Figure 2.2. To the left are the input-nodes and to the right the output-nodes. The neurons are the nodes gathered in between, in what is called the hidden layers. The output-nodes are mathematically equivalent to neurons. The number of hidden layers and the number of neurons in each hidden layer make up the structure of the ANN [21].

For this section some additional notation is required. Let N_{hl} be the number of hidden layers and let $l = \{0, 1, 2, ..., N_{hl} + 1\}$ indicate the layer, where $l = 0$ is the input layer and $l = N_{hl} + 1$ is the output layer. Also let layer l consist of M_l neurons and $n_{k,l}$ be the kth neuron of layer l. Assume two arbitrary neurons in adjacent layers l and $l + 1$, called neuron $n_{i,l}$ and neuron $n_{j,l+1}$, as shown in Figure 2.2 with $l = 1$. A link from unit $n_{i,l}$ to unit $n_{j,l+1}$ serves to propagate the activation $a_{i,l}$, from $n_{i,l}$ to $n_{j,l+1}$. Each of the links has a numeric weight, $w_{i,j,l}$, associated with it. This weight determines the sign and magnitude of the connection. Each layer has a dummy input, $a_{0,l}$, with a corresponding weight for each node in the layer, $w_{0,j,l}$, as well.

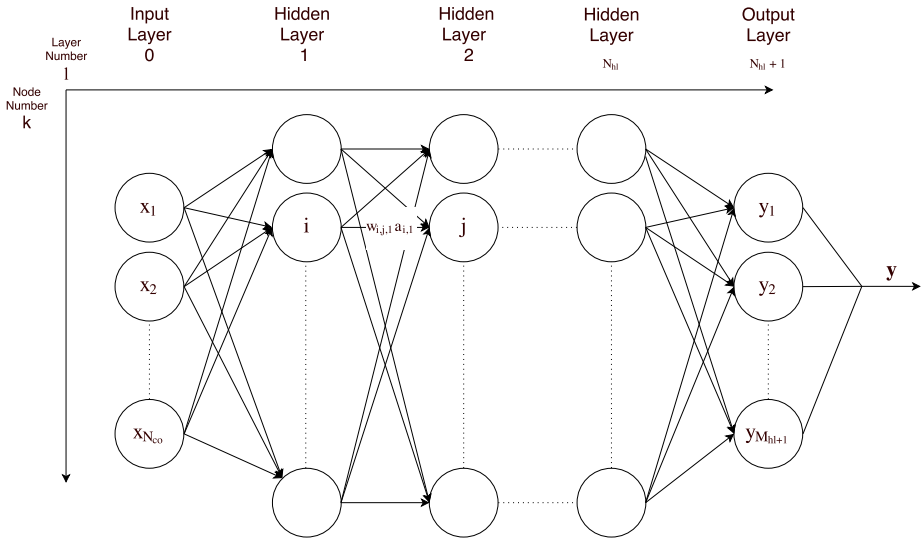

Figure 2.2: *An illustration of a neural network. Omitted are the bias inputs and their associated weights.*

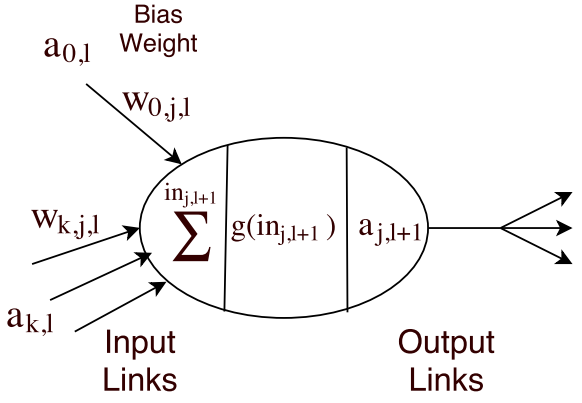

Figure 2.3: *A simple mathematical model of the neuron called j in Figure 2.2.*

The input to the node, $n_{j,l+1}$, is a weighted sum of the outputs from all the nodes in the previous layer

$$in_{j,l+1} = \sum_{k=0}^{M_l} w_{k,j,l} a_{k,l} \,. \tag{2.13}$$

The node then derives the output by applying the activation function, g, to this sum

$$a_{j,l+1} = g(in_{j,l+1}) = g(\sum_{k=0}^{M_l} w_{k,j,l} a_{k,l}) \,. \tag{2.14}$$

A graphical model of a neuron is provided in Figure 2.3.

The properties of the network are determined by three types of parameters.

- The interconnection pattern between the different layers of neurons; the number of hidden layers, the number of neurons etc.
- The weights of the interconnections, $w_{i,j,l}$, which are updated in the learning phase.
- The activation function, g, that converts a neuron's weighted input to an output.

The function g is sometimes a hard threshold, the neuron is then called a perceptron

$$g_\chi(z) = \chi_A(z) = \begin{cases} 1, & \text{if } z \in A. \\ 0, & \text{otherwise.} \end{cases} \tag{2.15}$$

Another common activation function is a logistic one

$$g_\sigma(z) = \frac{1}{1 + e^{-z}} \,. \tag{2.16}$$

In that case the neuron is called a sigmoid perceptron. The hyperbolic tangent function, tanh, is a rescaling of the logistic function, such that the output also ranges to negative numbers

$$g_h(z) = \tanh(z) = 2g_l(2z) - 1 = \frac{2}{1 + e^{-2z}} - 1 = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.17}$$

Learning and Prediction

The back-propagation algorithm, is a common method for training artificial neural networks. The idea is backward propagation of errors in combination with an optimization method, commonly the gradient descent method. Back-propagation consists of two phases which are cycled repeatedly, a propagation and a weight update. When the input is fed to the network, it is passed forward, one layer at

a time, until it reaches the end. The output of the network is then related to the labeled output, using a predetermined loss function, $V(\mathbf{y}, \mathbf{h}(\mathbf{x}, \mathbf{w}))$. This gives an error value for each of the neurons in the output layer. Starting from the output, these error values are propagated backwards. The error, associated with each neuron, will then roughly typify its addition to the initial output error. [15] [21]

The first phase consists of computing the loss function's gradient, with respect to the weights, using these errors, and passing it on to the optimization method. The second phase is updating the weights, in an attempt to minimize the loss function.

The following reasoning starts in the output layer. In order to further increase the readability the index k ranges over nodes in the output layer, j ranges over the nodes in the rightmost hidden layer and i ranges over the nodes in the second rightmost hidden layer. Using the squared Euclidian distance as loss function, $V(\mathbf{y}, \mathbf{h}(\mathbf{x}, \mathbf{w}) = |\mathbf{y} - \mathbf{h}(\mathbf{x}, \mathbf{w})|^2$, where the parameters to learn are the weights, $\theta = \mathbf{w}$, the gradient of the loss for any weight connecting to the output layer, $w_{j,k,N_{hl}}$, is

$$\begin{aligned}\frac{\partial}{\partial w_{j,k,N_{hl}}} V(\mathbf{y}, \mathbf{h}(\mathbf{x}, \mathbf{w}_{N_{hl}})) &= \frac{\partial}{\partial w_{j,k,N_{hl}}} |\mathbf{y} - \mathbf{h}(\mathbf{x}, \mathbf{w}_{N_{hl}})|^2 = \frac{\partial}{\partial w_{j,k,N_{hl}}} \sum_{k=0}^{M_{hl+1}} (y_k - a_{k,N_{hl}+1})^2 \\ &= \sum_{k=0}^{M_{hl+1}} \frac{\partial}{\partial w_{j,k,N_{hl}}} (y_k - a_{k,N_{hl}+1})^2 \end{aligned} \tag{2.18}$$

Here, as previously stated, k ranges over nodes in the output layer. The individual terms in the final summation corresponds to the gradient to the loss for the kth output, $V_k = (y_k - a_{k,N_{hl}+1})^2$. The gradient of this loss with respect to weights connecting the rightmost hidden layer with the output layer will be zero for all weights that do not connect to the kth output node. For the remaining weights, $w_{j,k,N_{hl}+1}$ it holds that

$$\begin{aligned}\frac{\partial}{\partial w_{j,k,N_{hl}}} V_k &= -2(y_k - a_{k,N_{hl}+1}) \frac{\partial a_{k,N_{hl}+1}}{\partial w_{j,k,N_{hl}}} = -2(y_k - a_{k,N_{hl}+1}) \frac{\partial g(in_{k,N_{hl}+1})}{\partial w_{j,k,N_{hl}}} \\ &= -2(y_k - a_{k,N_{hl}+1}) g'(in_{k,N_{hl}+1}) \frac{\partial in_{k,N_{hl}+1}}{\partial w_{j,k,N_{hl}}} \\ &= -2(y_k - a_{k,N_{hl}+1}) g'(in_{k,N_{hl}+1}) \frac{\partial}{\partial w_{j,k,N_{hl}}} (\sum_{j=0}^{M_{hl}} w_{j,k,N_{hl}} a_{j,N_{hl}}) \\ &= -2(y_k - a_{k,N_{hl}+1}) g'(in_{k,N_{hl}+1}) a_{j,N_{hl}} \stackrel{\Delta}{=} -2 a_{j,N_{hl}} \Delta_{k,N_{hl}+1} \end{aligned} \tag{2.19}$$

To obtain the gradient with respect to the weight connecting an arbitrary neuron in the second last hidden layer to a neuron in the last hidden layer, the chain rule needs to be reapplied and the activations expanded.

$$\begin{aligned}
\frac{\partial}{\partial w_{i,j,N_{hl}-1}} V_k &= -2(y_k - a_{k,N_{hl}+1}) \frac{\partial a_{k,N_{hl}+1}}{\partial w_{i,j,N_{hl}-1}} = -2(y_k - a_{k,N_{hl}+1}) \frac{\partial g(in_{k,N_{hl}+1})}{\partial w_{i,j,N_{hl}-1}} \\
&= -2(y_k - a_{k,N_{hl}+1}) g'(in_{k,N_{hl}+1}) \frac{\partial in_{k,N_{hl}+1}}{\partial w_{i,j,N_{hl}-1}} \\
&= -2\Delta_{k,N_{hl}+1} \frac{\partial}{\partial w_{i,j,N_{hl}-1}} (\sum_{j=0}^{M_{hl}} w_{j,k,N_{hl}} a_{j,N_{hl}}) \\
&= -2\Delta_{k,N_{hl}+1} w_{j,k,N_{hl}} \frac{\partial a_{j,N_{hl}}}{\partial w_{i,j,N_{hl}-1}} = -2\Delta_{k,N_{hl}+1} w_{j,k,N_{hl}} \frac{\partial g(in_{j,N_{hl}})}{\partial w_{i,j,N_{hl}-1}} \\
&= -2\Delta_{k,N_{hl}+1} w_{j,k,N_{hl}} g'(in_{j,N_{hl}}) \frac{\partial in_{j,N_{hl}}}{\partial w_{i,j,N_{hl}-1}} \\
&= -2\Delta_{k,N_{hl}+1} w_{j,k,N_{hl}} g'(in_{j,N_{hl}}) \frac{\partial}{\partial w_{i,j,N_{hl}-1}} (\sum_{i=0}^{M_{hl-1}} w_{i,j,N_{hl}-1} a_{i,N_{hl}-1}) \\
&= -2\Delta_{k,N_{hl}+1} w_{j,k,N_{hl}} g'(in_{j,N_{hl}}) a_{i,N_{hl}-1} \\
&= -2(y_k - a_{k,N_{hl}+1}) g'(in_{k,N_{hl}+1}) w_{j,k,N_{hl}} g'(in_{j,N_{hl}}) a_{i,N_{hl}-1} \qquad (2.20)
\end{aligned}$$

This process can be continued until the input layer is reached. Using gradient descent the learning process can be described as, update every weight, $w_{i,j,l}$, as

$$w_{i,j,l}^{n+1} = w_{i,j,l}^{n} - \alpha \frac{\partial}{\partial w_{i,j,l}} V_k, \qquad (2.21)$$

until convergence is achieved at the least possible loss. The step size, α, is usually called the learning rate. It can be a fixed constant or a decaying function of time as the learning phase proceeds. It is important to note that the back-propagation algorithm depends upon that the activation functions used by the neurons are differentiable [21].

Strengths and Weaknesses

The activation functions used by artificial neurons can be either linear or nonlinear. A network, made up of an interconnection of nonlinear neurons, is itself nonlinear. The usage of linear neurons limits the network to the capacity of regular linear regression. All three of the nonlinear activation functions, (2.15) (2.16) (2.17), ensures the important property that the entire network can represent a nonlinear function [21]. The threshold function (2.15) is nondifferentiable at $z = 0$, but the logistic function and the hyperbolic tangent function have the advantage of always being differentiable. Another benefit is that when it is operating in a dynamic environment, a neural network may be designed to update its weights in real time. This means that a neural network which was once trained to operate in a specific environment, easily can be retrained to deal with small

changes in the operating conditions. This adaptivity does however not necessarily lead to robustness. A neural network, using on-line learning, might tend to respond to spurious disturbances, which can cause a degradation in system performance [15].

2.2.3 Support Vector Machines

The support vector machine framework is currently the most popular approach for commercial usage [21]. Binary classification using support vector machines, SVMs, is done by finding a hyperplane, of dimension $N_{cov}-1$ in a N_{cov}-dimensional space, that separates the two classes. The hyperplane is also called the decision surface and the region in between the data points, of the two classes, is called the margin. There might be many hyperplanes that can classify the data. One reasonable choice is a hyperplane that separates the two classes of data, so that the margin is as large as possible. This is because having a larger margin reduces the problem of overfitting. The margin is, more formally, defined as the sum of distances, from the closest data points of both classes, to the hyperplane, $d_1 + d_2$. The hyperplane's equidistance from the two classes means $d_1 = d_2$. A simple illustration, in two dimensions, is provided in Figure 2.4.

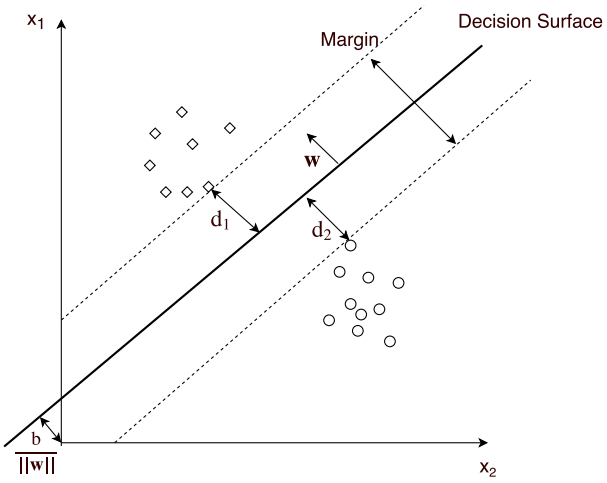

Figure 2.4: *A hyperplane fully separating two classes. Diamonds make up one class and circles make up another. x_1 and x_2 are the two covariates in a 2-dimensional space.*

Learning and Prediction

The goal with the learning is to find a hyperplane that maximizes the margin and that completely separates all the data points into two classes. If such a plane exists it is known as the maximum-margin separator [21]. A hyperplane, $\Pi(\mathbf{x})$, is defined as

$$\Pi(\mathbf{x}) \stackrel{\Delta}{=} \mathbf{w}\mathbf{x} + b = 0, \tag{2.22}$$

where $\frac{b}{\|\mathbf{w}\|}$ is the perpendicular distance, from the origin, to the plane and $\mathbf{w}$ is the plane's normal. The parameters to learn are $\mathbf{w}$ and b. Given outputs, labeled as $y_i \in \{1, -1\}\ \forall i$, can these parameters be adjusted, so that the plane fulfills the constraints

$$\begin{cases} \mathbf{w}\mathbf{x}_i + b \geq 1, & \text{if } y_i = 1. \\ \mathbf{w}\mathbf{x}_i + b \leq -1, & \text{if } y_i = -1. \end{cases} \tag{2.23}$$

In Figure 2.4 do these constraints represents the two parallel bounding hyperplanes, marked by dotted lines. Vector geometry shows that the margin is equal to $\frac{1}{\|\mathbf{w}\|}$ and maximizing $\frac{1}{\|\mathbf{w}\|}$, subject to the constraints (2.23), is equivalent to minimizing $\|\mathbf{w}\|$. This in turn is equivalent to finding the minimum to $\frac{1}{2}\|\mathbf{w}\|^2$. Consequently the maximum-margin separator is found by solving the optimization problem

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \end{aligned} \tag{2.24}$$

where the two constraints in (2.23) have been combined to one inequality. The optimization problem, (2.24), assumes that there exists a maximum-margin separator, i.e. that the dataset given is separable by a plane. In order to extend SVM to cases in which the data is not linearly separable, slack variables, ξ_i, are introduced to relax the constraints

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{2.25}$$

Here $C > 0$ is a tunable penalty parameter for the error term. The separable case corresponds to $C = \infty$. The optimization problem, (2.25), is quadratic with linear inequality constraints, which means that it is a convex optimization problem. The quadratic programming solution can be described using Lagrange multipliers [13]. The primal Lagrange function is

$$L_p = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i. \tag{2.26}$$

Minimizing this with respect to $\mathbf{w}$, b and ξ results in a set of derivatives. Setting these derivatives to zero gives

$$\mathbf{w} = \sum_{i=1}^{N} \mathbf{x}_i y_i \alpha_i, \tag{2.27}$$

$$0 = \sum_{i=1}^{N} y_i \alpha_i, \tag{2.28}$$

$$\alpha_i = C - \mu_i, \tag{2.29}$$

together with positivity constraints on ξ_i, α_i and μ_i. Substituting (2.27) into (2.26) gives the dual objective function

$$L_d = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j). \tag{2.30}$$

In conclusion the dual optimization problem is

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j) \\ \text{s.t.} \quad & 0 = \sum_{i=1}^{N} y_i \alpha_i, \\ & 0 \leq \alpha_i \leq C. \end{aligned} \tag{2.31}$$

In order to find the sought after parameters, $\mathbf{w}$ and b, the remaining Karush–Kuhn–Tucker conditions

$$\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i)] = 0, \tag{2.32}$$

$$\mu_i \xi_i = 0, \tag{2.33}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) - (1 - \xi_i) \geq 0, \tag{2.34}$$

are used, alongside (2.27)-(2.29). Together the optimization problem and the constraints, (2.27)-(2.34), uniquely characterize the solution to the primal problem (2.25). There exist good software packages to solve the quadratic programming problem [21]. The optimal value for C can be estimated by cross-validation, which is further described in Section 3.1.4.

A fixed, nonlinear, covariate-space transformation is denoted $\phi(\mathbf{x})$. The inner product of the nonlinearities of two covariate vectors,

$$K(\mathbf{x}_i, \mathbf{x}_j) \triangleq \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \tag{2.35}$$

is called the kernel function [21]. It is often not expected to find a linear separator in the input space of the original covariate vector, $\mathbf{x}$, but a linear separator in a higher dimension can be found by replacing $\mathbf{x}_i^T \mathbf{x}_j$ with $K(\mathbf{x}_i, \mathbf{x}_j)$ in (2.30). By applying this so-called kernel trick, the resulting linear separators can correspond

to arbitrary nonlinear decision surfaces, when mapped back to the original input space. For this to work $K(\mathbf{x}_i, \mathbf{x}_j)$ should be a symmetric, positive semi-definite function [13]. Substituting $\mathbf{x}_i^T \mathbf{x}_j$ with $K(\mathbf{x}_i, \mathbf{x}_j)$ in (2.30) corresponds to solving the primal problem

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{2.36}$$

The function describing the hyperplane, which includes the nonlinear transformation of the covariate vector $\phi(\mathbf{x})$, can be rewritten, using (2.27), as

$$\mathbf{w}^T\phi(\mathbf{x}_j) + b = (\sum_{i=1}^{N}[\phi(\mathbf{x}_i)y_i\alpha_i])^T\phi(\mathbf{x}_j) + b = \sum_{i=1}^{N}[y_i\alpha_i\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)] + b. \tag{2.37}$$

This means that both the primal and the dual problem involve $\phi(\mathbf{x})$ only through inner products. Therefore the transformation, $\phi(\mathbf{x})$, needs not be specified at all and only knowledge of the kernel function, $K(\mathbf{x}_i, \mathbf{x}_j)$, is required. The most simple kernel function is the linear one that was used in (2.31)

$$K_l(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j. \tag{2.38}$$

A slightly more advanced alternative is the polynomial kernel

$$K_p(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j)^\gamma, \ \gamma > 0. \tag{2.39}$$

Another popular choice is the Gaussian kernel

$$K_G(\mathbf{x}_i, \mathbf{x}_j) = e^{(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)}, \ \gamma > 0. \tag{2.40}$$

Here γ is a kernel parameter that is either predetermined or found by cross validation. Prediction is done by reversing the logic of (2.23) and using the equation for the hyperplane in (2.37)

$$\begin{cases} \hat{y} = 1, & \text{if } \sum_{i=1}^{N}[y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}_j)] + b \geq 0. \\ \hat{y} = -1, & \text{if } \sum_{i=1}^{N}[y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}_j)] + b \leq 0. \end{cases} \tag{2.41}$$

Strengths and Weaknesses

Support vector machines are nonparametric and, when using them, it is potentially required to store all the training examples, see Equation (2.41). In practice only a small fraction is, most often, actually retained [21]. All points of data that lie in the hyperplanes in (2.41) are the support vectors, hence the name of the method [5]. For SVMs, the support vectors are the critical elements of the training set. This is because if all other training points were removed and the training

procedure was repeated, the same separating hyperplane would be found. For this reason SVMs can be said to combine the advantages of nonparametric and parametric models. They have the flexibility to represent complex functions but are still quite resistant to overfitting [21].

Another strength is the ability to embed the data into a higher-dimensional space. This is done using the aforementioned kernel trick. The idea is that non-linear surfaces in a low dimension, can be approximated with equivalent linear ones in a higher dimension.

2.2.4 Gaussian Processes

As mentioned in Section 2.1.1, in Bayesian learning predictions are made by using all the hypothesis functions, weighted by their probabilities, rather than solely using the best one. One approach to do this, is to give a prior probability to every possible hypothesis function, $h(\mathbf{x})$, where higher probabilities are given to ones that are considered to be more likely. This can for example mean functions that are simpler or smoother. A common prior over functions is the Gaussian process, GP, which is a class of stochastic processes that have proved very successful to do so [27].

Whereas a probability distribution describes random variables which are scalars or vectors, a stochastic process governs the statistical properties of functions and a Gaussian process is a generalization of the Gaussian probability distribution.

A Gaussian process can be defined as a set of stochastic variables which belong to a joint Gaussian distribution

$$\Pr(h_i, h_j, h_k, ...) = \left(\begin{bmatrix} m(\mathbf{x}_i) \\ m(\mathbf{x}_j) \\ m(\mathbf{x}_k) \\ \vdots \end{bmatrix}, \begin{bmatrix} K(\mathbf{x}_i, \mathbf{x}_i) & K(\mathbf{x}_i, \mathbf{x}_j) & K(\mathbf{x}_i, \mathbf{x}_k) & \\ K(\mathbf{x}_j, \mathbf{x}_i) & K(\mathbf{x}_j, \mathbf{x}_j) & K(\mathbf{x}_j, \mathbf{x}_k) & \\ K(\mathbf{x}_k, \mathbf{x}_i) & K(\mathbf{x}_k, \mathbf{x}_j) & K(\mathbf{x}_i, \mathbf{x}_k) & \\ & & & \ddots \end{bmatrix} \right). \tag{2.42}$$

As notation for a function that follows a Gaussian process,

$$h(\mathbf{x}_i) \sim \mathcal{GP}(\mathbf{m}(\mathbf{x}), \mathbf{K}(\mathbf{x}, \mathbf{x}')), \tag{2.43}$$

is used. In order to fully specify the GP only the mean function $\mathbf{m}(\mathbf{x})$ and covariance function $\mathbf{K}(\mathbf{x}, \mathbf{x}')$ are required [14]. The covariance function is written with the notation of a capital, K, because it is essentially the kernel of a GP and determines the shape of its prior and posterior [14].

The values of a hypothesis function at a particular input location $h(\mathbf{x}_i)$ is denoted by the random variable h_i. When using a Gaussian process for machine learning, a finite set of points are selected, $\mathbf{x} \in \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, at which to evaluate a hypothesis function. Namely the training data set. The values of a hypothesis function at those locations, $\mathbf{h} \triangleq \left[h_1, ..., h_N\right]$ is a vector of stochastic variables. It follows a Gaussian distribution

$$\Pr(\mathbf{h}|\mathbf{X}) = \mathcal{N}(\mathbf{m}(\mathbf{x}), \mathbf{K}(\mathbf{x}, \mathbf{x'})), \tag{2.44}$$

where $\mathbf{m}(\mathbf{X})$ and $\mathbf{K}(\mathbf{X})$ are the mean vector and covariance matrix defined in the same way as in Equation 2.42.

Learning and Prediction

The distribution $\Pr(\mathbf{h}|\mathbf{Y},\mathbf{X})$ represents the posterior over the hypothesis function $h(x)$ at all locations in the training set. This can be useful in itself, but for prediction purposes it is the values of $h(x)$ at other locations in the input space that are most interesting. I.e., the predictive distribution of $h_* = \mathrm{h}(\mathbf{x}_*)$ at a new location $\mathbf{x}_*$,

$$\Pr(h_*|\mathbf{X}_*,\mathbf{Y},\mathbf{X}). \tag{2.45}$$

This can be achieved by integrating out h

$$\Pr(h_*|\mathbf{X}_*,\mathbf{Y},\mathbf{X}) = \int \Pr(h_*,\mathbf{h}|\mathbf{X}_*,\mathbf{Y},\mathbf{X})d\mathbf{h} = \int \Pr(h_*|\mathbf{h},\mathbf{X}_*,\mathbf{X})\Pr(\mathbf{h}|\mathbf{Y},\mathbf{X})d\mathbf{h}, \tag{2.46}$$

where the first factor in the second integral is always Gaussian. This is a result from the Gaussian process prior which links all possible values of h and h_* to a joint normal distribution. The second term, $p(\mathbf{h}|\mathbf{y},\mathbf{x})$, is the posterior of h. Bayes' theorem can be applied in the conventional manner to obtain a posterior over the hypothesis function at all locations where training data is available.

$$\Pr(\mathbf{h}|\mathbf{Y},\mathbf{X}) = \frac{\Pr(\mathbf{Y}|\mathbf{h})\Pr(\mathbf{h}|\mathbf{x})}{\Pr(\mathbf{Y}|\mathbf{X})} = \frac{\Pr(\mathbf{Y}|\mathbf{h})\mathcal{N}(\mathbf{h}|m(\mathbf{X}),K(\mathbf{X}))}{\Pr(\mathbf{Y}|\mathbf{X})} \tag{2.47}$$

In the particular case where the likelihood has the form of a given distribution, this posterior can be computed analytically. This is the case in Gaussian process regression when additive Gaussian noise is considered. However, for arbitrary likelihood functions the posterior will not necessarily be Gaussian, which is the case in classification. For arbitrary likelihoods it is necessary to use approximation methods. For binary classification the integral is one-dimensional and in that case a simple, numerical, technique is most often adequate [14]. The object of central importance, for all the approximation methods, is the posterior distribution $\Pr(h|\mathbf{X},\mathbf{Y})$. One common approximation methods is the Laplace method.

By doing a second order Taylor expansion of the logarithm of the posterior $\log[\Pr(h|\mathbf{x},\mathbf{y})]$ around its maximum a Gaussian approximation, $q(h|\mathbf{x},\mathbf{y})$, is

$$q(h|\mathbf{X},\mathbf{Y}) = \mathcal{N}(h|\hat{h},A^{-1}) \propto e^{-\frac{1}{2}(h-\hat{h})^T A(h-\hat{h})}. \tag{2.48}$$

Here $A = -\nabla\nabla\Pr((h|\mathbf{X},\mathbf{Y}))|_{h=\hat{h}}$ is the Hessian of the negative log posterior at the point point $\hat{h} = \mathrm{argmax}_h \Pr(h|\mathbf{X},\mathbf{Y})$. After finding the maximum of h, using Newton's method, the Hessian is available analytically by differentiation. This is the main idea of the Laplace approximation method. The method is described thoroughly in [27].

After having found an approximation for the posterior it is possible to marginalize the unknown values of the hypothesis function and obtain a marginal likelihood, $\Pr(\mathbf{y}|\mathbf{x})$. This marginal likelihood is tractable [14]. Maximizing the

marginal likelihood with respect to the mean and covariance functions provides a practical way to perform Bayesian model selection. The characteristics of a Gaussian process model can be controlled by writing the mean and covariance functions in terms of what is called hyperparameters. It is these hyperparameters that are estimated during training. As in the case with SVMs there exist many common covariance functions. The ones given by Equation (2.38)-(2.40) work for Gaussian processes as well.

In binary classification the basic idea behind Gaussian process prediction is to place a GP prior over the function h and to squeeze this through the logistic function, (2.16),

$$\pi(\mathbf{x}) \stackrel{\Delta}{=} \Pr(y_x = +1) = g_\sigma(h(\mathbf{x})). \tag{2.49}$$

π is a deterministic function of h and because $h(\mathbf{x})$ is stochastic, so is π. The purpose of h is solely to give a convenient formulation of the model and the values of $h(\mathbf{x})$ are not of interest. The only interest is the value of the test case $\pi(\mathbf{x}_*)$ [27]. This way the output from a regression model, that initially can lie in the domain $[-\infty, \infty]$, is put into the range $[0, 1]$, which guarantees a valid probabilistic interpretation.

The predicted class is determined by using the hypothesis function that provides a good fit for all the observed data

$$\Pr(y_* = +1) = \int \pi(\mathbf{x}) \Pr(h_*|\mathbf{y}, \mathbf{x}_*) d h_*. \tag{2.50}$$

Because the classification is binary, the probability of a negative class is

$$\Pr(y_* = -1) = 1 - \Pr(y_* = +1). \tag{2.51}$$

Strengths and Weaknesses

The Gaussian process classifier developed in this section is discriminative and nonparametric. This means that it is required to store all the training examples. In the prediction process it is required to invert a $N \times N$ matrix which makes the basic complexity $O(N^3)$ [27]. GPs do have the flexibility to represent complex functions but it is a major set back that the prediction is computationally difficult.

An issue with the Laplace approximation is that the Hessian evaluated at $\hat{h}$, can give a poor approximation of the posterior's true shape. The peak could either be broader or narrower than the Hessian indicates [27].

2.3 Evaluation of Binary Classifiers

There are many metrics that can be used to measure the performance of a classifier. Different scientific fields also have different preferences for specific metrics due to having differing goals.

2.3.1 Contingency Table

In the field of machine learning, specifically the problem of statistical classification, a contingency table, also known as a confusion matrix, is a specific table layout that allows visualization of the performance of a classifier. Given a classifier and some validation data the predicted class can be compared with the true class. A binary classifier with inputs being mapped to one of two discrete classes will result in four outcomes depending on the predicted class and actual class, as illustrated in Figure 2.5. In the figure the notation {p,n} and {P,N} are used for positive and negative class labels, for the actual and predicted class respectively. Each row of the table represents the outcomes in a predicted class while each column represents the outcomes in an actual class. This is the basis for many common metrics. [10]

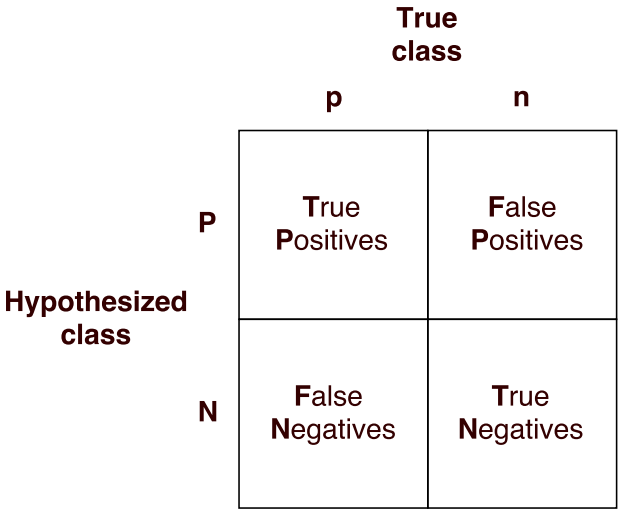

Figure 2.5: *A Contingency table.*

From the contingency table, four useful quantities can be calculated. These are the false positive rate, FPR, the true positive rate, TPR, the precision and the

accuracy.

$$\text{FPR} = \frac{\text{False positives}}{\text{Total negatives}} = \frac{\text{False positives}}{\text{False positives + True negatives}},$$
$$\text{TPR} = \frac{\text{True positives}}{\text{Total positives}} = \frac{\text{True positives}}{\text{True positives + False negatives}},$$
$$\text{precision} = \frac{\text{True positives}}{\text{True positives + False positives}},$$
$$\text{accuracy} = \frac{\text{True positives + True negatives}}{\text{Total positives + Total negatives}}.$$

2.3.2 Receiver Operating Characteristics

A receiver operating characteristics (ROC) graph is another technique for selecting classifiers based on their performance. ROC graphs are conceptually simple, especially when considering binary classification problems.

ROC graphs are two-dimensional graphs in which TPR is plotted on the y-axis and the FPR is plotted on the x-axis. This results in an illustration of relative trade-offs between how many correct results that occur among the positive samples and how many incorrect results that occur among the negative samples. The diagonal line, $y = x$, represents the strategy of randomly guessing a class. For example, if a classifier randomly guesses the positive class half the time, it can be expected to get half the positives and half the negatives correct [10]. A basic ROC graph is given in Figure 2.6.

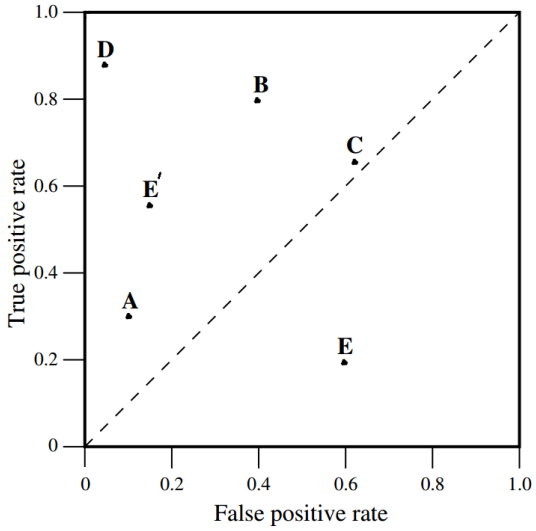

Figure 2.6: *A basic ROC graph showing the performance of some example classifiers.*

A couple of coordinates in the ROC space are notable. The bottom left point, $(0, 0)$ represents the strategy of never issuing a positive classification; such a classifier commits no false positive errors but also gains no true positives. In the case of prediction of the preceding vehicle's lane switch intentions this would be equal to a classifier that never calls for any lane switch at all. The opposite point, $(1, 1)$ represents the strategy, of unconditionally issuing positive classifications. The point $(0, 1)$ stands for perfect classification and a classifier that ends up in the lower right corner performs worse than a model, constantly, guessing the outcome.

By declaring every positive prediction as negative and vice versa it is possible to invert the output of a classifier. This means that its true positive classifications become false negative mistakes, and its false positives become true negatives. In the ROC graph this corresponds to having its position mirrored in the $y = x$ line, see the point E that is mirrored to E' in Figure 2.6.

2.4 Multiclass Classification

In opposite to binary classification, multiclass or multinomial classification is the problem of classifying instances into one of more than two classes. There are some classification algorithms that can do this by default, for example classification with ANNs. The usage of neural networks provides a straightforward extension to the multiclass problem. Instead of having one neuron in the output layer, with binary output, multiple neurons can be used. Other classification algorithms, like SVMs, are by nature binary. These can, however, be turned into multinomial classifiers by using a variety of strategies.

2.4.1 One-Versus-the-Rest

One idea would be to build a N_c-class classifier by combining a number of binary ones and have each of them classify the data as positive, if it belongs to that class or negative if it does not. This is known as one-versus-the-rest classification. In cases where the classifier produces a real-valued confidence score for its decision, like a probability, it is possible to make a prediction by applying all classifiers to an unseen sample and predicting the label for which the corresponding classifier, f_c reports the highest score,

$$\hat{y} = \underset{c \in \{1,\ldots,N_c\}}{\operatorname{argmax}} f_c(\mathbf{x}). \tag{2.52}$$

Some classifiers only outputs a class label. This leads to some difficulties with ambiguities [3]. Figure 2.7 shows a case where two binary classifiers are used to separate three classes and the one-versus-the-rest strategy makes a region of the input space ambiguously classified.

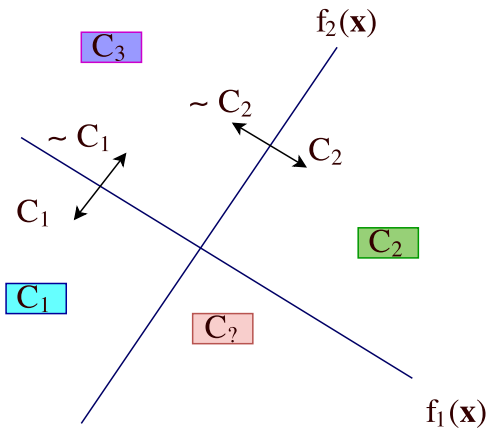

Figure 2.7: *One-Versus-the-Rest classification leads to ambiguous regions, shown in red.*

2.4.2 One-Versus-One

An alternative approach is to introduce one classifier for each possible pair of classes, $f_{c_i,c_j}(\mathbf{x})$. This is known as one-versus-one classification and requires the training of $N_c \frac{(N_c-1)}{2}$ individual classifiers. Each point is then classified according to a majority vote between the different classifiers as illustrated in Figure 2.8. As seen in the figure, this too runs into the problem of ambiguous regions.

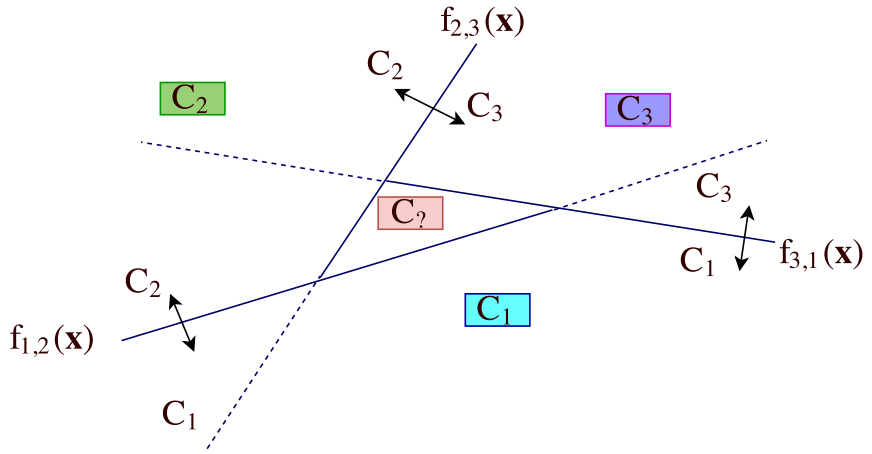

Figure 2.8: *One-Versus-One classification leads to ambiguous regions, shown in red.*

3

Implementation

This chapter describes the steps required in order to actualize and implement the theory from Chapter 2 in practice. The first sections are devoted to describe how sensor data was acquired. The chapter also aims to describe how sensor data was refined in order to obtain the highest possible accuracy for the classifiers.

3.1 Acquiring Training Data

All sensors in a Scania truck are connected to the built-in controller area network (CAN). During several thousand kilometers of field trips, these sensor signals have been stored for analysis purposes. Together with CAN data, there exists recorded video streams of the scenario in front of the vehicle for some of the trips. The intention of these video streams is to easily verify the occurrence of a specific maneuver in the logged data. To train and test the different algorithms, specific data was required. From the large CAN logs were therefore smaller subsets of data extracted defining situations of interest.

As mentioned in Section 1.3 the thesis was constrained to only include cases when the tracking was persistently substantial. This was done because following the preceding vehicle autonomously, does require that estimates of the preceding vehicle's position and motion are available anyway. If the tracking was not to be working the in this thesis developed system function, would therefore not serve a meaningful purpose.

Collection of training data was a quite cumbersome process because driving maneuvers happen on a very large time scale. Maneuvers are also often combined or aborted which makes collection of clean data difficult. Some of the choices made in the process are explained in the following sections.

3.1.1 Defining a lane change

Ideally the start of a lane change action would probably be defined as the moment at which a vehicle starts to steer towards the destination, i.e. stops being parallel to the lane and the end as the time at which the vehicle, once again, is parallel to the lane. Since a very small set of signals describing the state of the preceding vehicle was available it was, however, hard to determine these start and end points. What was more achievable was to determine the point at which the preceding vehicle and the road markers, indicating the lane border, was at the same position. This point can be assumed to take place halfway through the lane change.

Since this thesis aimed to predict the lane change early enough for the system to trigger a new function, the data from the second half of the manoeuvre was considered redundant. This is because indicating a lane change that late would not provide the system with enough time to respond accordingly even if the call was correct and training on that data would therefore be ineffective.

It should perhaps be emphasized that some authors of related works, like [28], have suggested the inclusion of preparatory actions before the driver actually commence the actual lane-change in the lane-change time. Determining the start of preparatory actions without access to eye- or head trackers is however essentially impossible and this was therefore excluded. Given the above stated ideal definition of a lane change, the average lane change takes roughly five seconds [29].

3.1.2 Extracting interesting subsets of data

A delimitation of the thesis was that only lane switches that occur on highways, including entrance and exit ramps, were to be dealt with. One easy way to sort out data that did not fulfill this requirement was to eliminate all scenarios where the vehicle speed was less than 70 kilometers per hour. In addition to that, only data segments from driving when there was a preceding vehicle present were considered. In order to receive comprehensive training data that contained all the information required for detection of lane changes, only occasions where a preceding vehicle was present for more than 25 seconds were kept. Because most of the driving, during data logging, was done on highways, a substantial part of the data did meet these limitations. The amount of scenarios where the occasion of vehicle following ended with the preceding vehicle performing a lane change, or leaving the highway, was however significantly smaller. This criteria was fulfilled for about 5 percent of all the previously obtained data segments. It was considered to also use scenarios of vehicle following that did not end with a lane change for training, in order to teach the model what to classify as a true negative, but even if these scenarios were disregarded the portion of data, corresponding to no lane switch, amounted to more than 98 percent of the total training data.

3.1.3 Preprocessing

Under all these criteria the signal values were stored at a sampling rate of 100 Hz. Most of the considered signals had a lot of high frequency components that originated from vibrations, vehicle motion and the sensors themselves. These were decided to be filtered out to highlight data of interest. A lowpass filter of Butterworth type was therefore added to dampen frequencies above a cutoff frequency of 6 Hz. This did introduce a delay for the filtered signals, of roughly one tenth of a second but the benefits of having a smooth signal was considered more important. The cutoff frequency was carefully selected in order to not remove any of the sought after trends and still make it possible for the classifier to capture significant changes in data. To not have any signal influencing the classifier more than another, as a final step before being presented as input, all the signals were also scaled down to an interval between -1 and 1.

3.1.4 Partitioning of data

To get an accurate evaluation of a classifier, it needs to be presented some examples it has not yet seen. The data is therefore divided into three parts called the training set, the test set and the validation set. The training set is used to fit the models and the test set is used to estimate prediction error for model selection. The validation set is the set of data on which the classifier is evaluated. The validation is meant to be a simulation of what would happen if the classifier was implemented in reality. Ideally, the validation set should be brought out only at the end of the data analysis [13].

Despite the high amount of stored driving data, the subset viable for training was limited. A problem with this was that there were not enough data available

to partition it into separate conventional sets, as in Figure 3.1, without losing significant modelling capability. If the test set is big, a lot of training data goes to waste and if the test set is small, there is a statistical risk of getting a poor estimate of the actual accuracy.

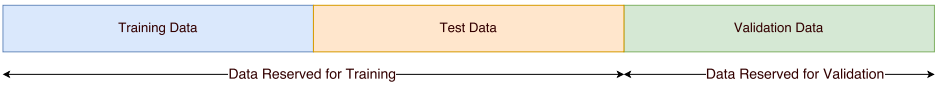

Figure 3.1: *Conventional partitioning of data, e.g. one predetermined set is held out for testing.*

In cases like this, a fair way to properly estimate model prediction performance is to use cross-validation. One round of cross-validation involves partitioning the data into complementary subsets, performing the analysis on one subset, and testing the analysis on the other. Multiple rounds of cross-validation can be performed, using different partitions, in order to reduce variability. The test results can then be averaged over the rounds. This way data reserved for training can be utilized more efficiently, because no data needs to be held out for the sole purpose of testing, see Figure 3.2.

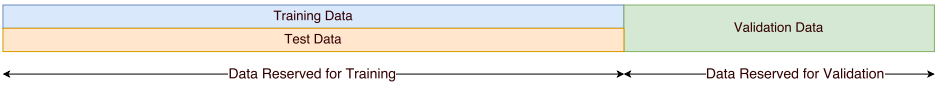

Figure 3.2: *When using cross-validation, a bigger portion of the data can be used for training.*

Two types of cross-validation can be distinguished, exhaustive and non-exhaustive. Methods are exhaustive if the goal is to learn and test on all possible ways to divide the original sample into a training and a testing set. An example of an exhaustive method is Leave-p-out. This method involves using p observations as the testing set and the remaining observations for training. This is repeated on all ways to cut the original data set. The simplest case, with $p = 1$, is called Leave-one-out. Leave-p-out can be computationally intractable, even when p is small, because it requires training and testing of the model $\binom{N}{p}$ times [2].

Non-exhaustive cross validation methods do not compute all ways of splitting the original data set. The idea of k-fold-cross-validation, which is a non-exhaustive method, is to first split the data into k equal subsets. k rounds of learning are then performed, on each of which a different $\frac{1}{k}$ of the data is held out for testing and the remaining examples are used for training. Common choices of k are 5 and 10, which are enough to statistically give a good estimate, at the cost of 5 to 10 times longer computation time [21]. The most extreme case of k-fold-cross-validation is having k equal to the size of the training set, $k = N$. This is exactly equal to the aforementioned leave-one-out method. The k-fold-cross-validation procedure is illustrated in Figure 3.3.

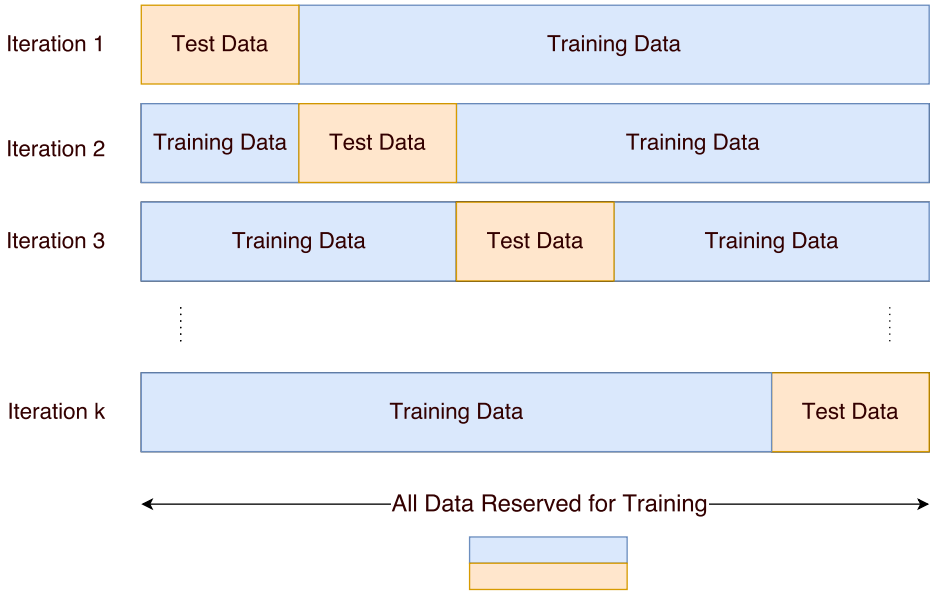

Figure 3.3: *The k-fold-cross-validation procedure.*

3.2 Transformation of Sensor Data

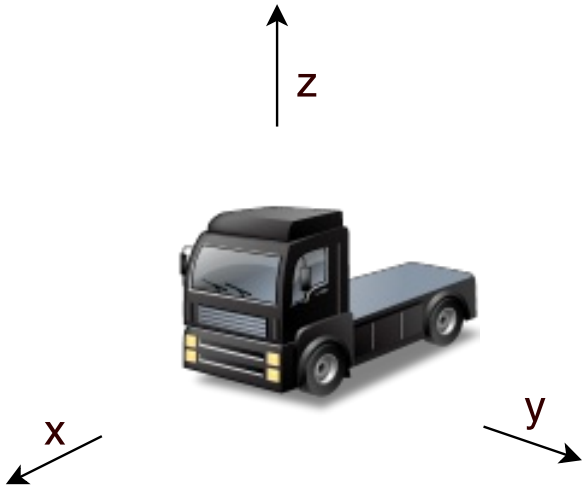

Figure 3.4: *Definition of the longitudinal axis (x), lateral axis (y) and vertical axis (z).*

A multitude of sensor data is available through already existing sensor fusion and the signals used in this thesis are summarized in Table 3.1. The original signals can be combined and extended, using knowledge about physics, to better and more compactly present input to the classifier. The derivation of these enhanced signals are provided in this section.

Category	Signal name	Notation	Unit
Host vehicle dynamics	Longitudinal velocity	$v_{x,host}$	$\frac{m}{s}$
	Yaw rate	ω	$\frac{rad}{s}$
Preceeding vehicle dynamics	Longitudinal velocity (relative host)	$v_{x,rel}$	$\frac{m}{s}$
	Lateral velocity (relative host)	$v_{y,rel}$	$\frac{m}{s}$
Environment data	Longitudinal dist. to prec. vehicle	D_x	m
	Lateral dist. to prec. vehicle	D_y	m
	Road curvature	κ	$\frac{1}{m}$
	Lane marker data	M, K, A, B	-

Table 3.1: *Used sensor data.*

3.2.1 Mimimum distance to lane edge

Ways to predict the intention of the vehicle in front of oneself were primarily sought in the case where no clear lane-markers were available. In order to be

able to evaluate the results it can, however, be of interest to compare the outcome with one where lane-makers were used.

The lane-marker data is logged as a third degree polynomial,

$$y = M + Kx + Ax^3 + Bx^2. \tag{3.1}$$

Here x and y are coordinates in the system presented in Figure 3.4, where the the host vehicle makes up the origin and a set of values for the parameters, M, K, A and B is stored for each sample. During ordinary lane-following two lane-markers are usually available, one on each side and given the position of the preceding vehicle, (D_x, D_y), it is possible to compute the distances to the lane edges

$$\Delta_{left} = |M_{left} + K_{left}D_x + A_{left}{D_x}^3 + B_{left}{D_x}^2 - D_y|,$$
$$\Delta_{right} = |M_{right} + K_{right}D_x + A_{right}{D_x}^3 + B_{right}{D_x}^2 - D_y|.$$

Just the smallest one of these two is relevant for the classifier and was used as input

$$\Delta_{lane} = min\{\Delta_{left}, \Delta_{right}\}. \tag{3.2}$$

3.2.2 Deviation from expected lateral velocity

Using a road map and the GPS it is possible to extract the current curvature of the road. The curvature is provided as the reciprocal of the radius to the unique circle or line which most closely approximates the curve at a given point, P, see Figure 3.5. For a straight line the radius will lean to infinity and consequently the curvature will go towards zero.

$$\kappa = \frac{1}{R},$$

The absolute velocity of the preceding vehicle together with this radius is used to estimate the lateral velocity required to follow the road. Deviation from this would indicate a lane-switch.

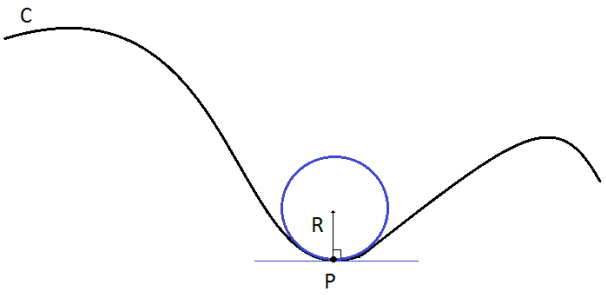

Figure 3.5: *C is the road, P is the position provided by the GPS, and R is the radius.*

Given the current position of a vehicle, P_1 and its absolute velocity, v, a future position, P_2, t seconds ahead can be estimated by assuming that the current curvature will persist. As seen in Figure 3.6, the lateral deviation, $d(t)$, at this point is given by trigonometry as

$$d(t) = R - x(t) = R(1 - \cos(\theta(t))). \tag{3.3}$$

Due to the laws of circular motion the vehicle's angular velocity is

$$\omega = \frac{v}{R},$$

which gives

$$\theta(t) = \omega t = \frac{vt}{R}. \tag{3.4}$$

The expected lateral velocity in the next sample is given by

$$v_{y,expected} = \frac{d(T_s)}{T_s}, \tag{3.5}$$

and combining (3.3), (3.4) and (3.5) gives

$$v_{y,expected} = \frac{R(1 - \cos(\frac{v}{R}T_s))}{T_s} = \frac{1 - \cos(\kappa v T_s)}{\kappa T_s}. \tag{3.6}$$

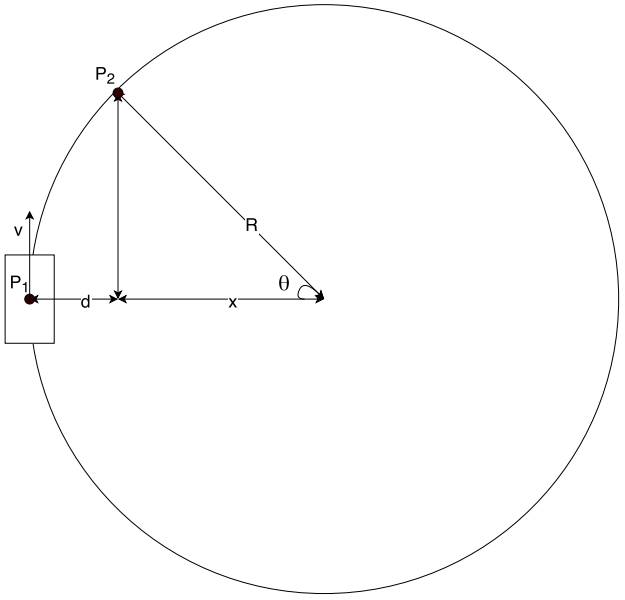

Figure 3.6: *A vehicle following the circle corresponding to the current road curvature.*

In order to compare the estimated lateral velocity of the preceding vehicle, which is given relative to the host vehicle, a correction has to be made regarding the experienced lateral velocity, caused by the host vehicle turning. Using the yaw rate of the host vehicle, ω, the longitudinal distance between the two vehicles, D_x and the same law of circular motion as before, this induced velocity is given by,

$$v_{y,induced} = -\omega D_x. \tag{3.7}$$

In conclusion the deviation from expected lateral velocity, for the preceding vehicle, ΔE_{v_y} is

$$\begin{aligned} \Delta E_{v_y} &= |v_{y,rel} - v_{y,induced} - v_{y,expected}| \\ &= |v_{y,rel} + \omega D_x - \frac{1 - cos(\kappa(v_{x,host} + v_{x,rel})T_s)}{\kappa T_s}|. \end{aligned} \tag{3.8}$$

No concern is given whether the deviation is to the left or to the right, since lane-switches in both directions are treated the same way. In Figure 3.7 the signal ΔE_{v_y} is plotted in blue together with two lane switches in orange. It can be seen that the signal is quite noisy. Alongside other covariates it can however be resourceful.

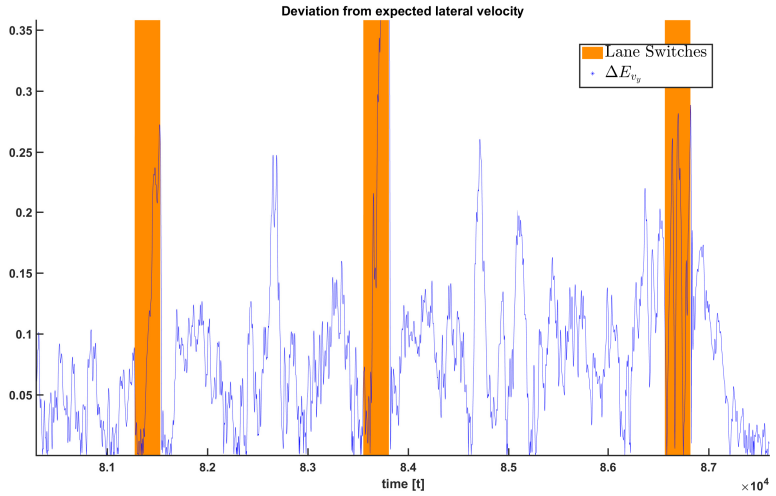

Figure 3.7: *ΔE_{v_y}, scaled down to an interval between 0 and 1, is plotted in blue together with three lane switches in orange. The signal is noisy but does increase in amplitude for all of the lane switches.*

Artificial road curvature

The road curvature from GPS is not available in every truck. In the cases where it is missing the signal can be artificially manufactured using the lane-markers. Using the definition in [26], where s is the measure of arc-length along a curve and ψ is the tangential angle shown in Figure 3.8, curvature is the magnitude of the rate of change of ψ with respect to the distance moved along the curve. This means that

$$\kappa = \frac{d\psi}{ds}.$$

Since the curve, according to Equation (3.1), is given on the form $y = f(x)$ it is noted that

$$\frac{d\psi}{ds} = \frac{d\psi}{dx}\frac{dx}{ds} = \frac{(\frac{d\psi}{dx})}{(\frac{ds}{dx})}. \tag{3.9}$$

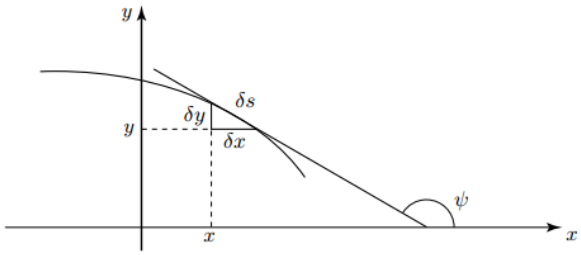

Figure 3.8: *The tangent line makes an angle ψ with the positive x-axis. Small increments in x and y have been denoted by δx and δy, respectively.*

Recognition of the triangle in Figure 3.8 and Pythagoras' theorem gives

$$\delta s^2 = \delta x^2 + \delta y^2 \Leftrightarrow (\frac{\delta s}{\delta x})^2 = 1 + (\frac{\delta y}{\delta x})^2 \Rightarrow \frac{\delta s}{\delta x} = \sqrt{1 + (\frac{\delta y}{\delta x})^2}. \tag{3.10}$$

As the increments get smaller this equals the derivative

$$\frac{ds}{dx} = \sqrt{1 + (\frac{df}{dx})^2} = (1 + f'(x)^2)^{\frac{1}{2}}. \tag{3.11}$$

It is also notable that, as seen in Figure 3.8

$$\frac{df}{dx} = \frac{\delta y}{\delta x} = -\tan(\pi - \psi) = \tan(\psi). \tag{3.12}$$

Differentiating this gives

$$\frac{d^2 f}{dx^2} = (1 + \tan^2(\psi))\frac{d\psi}{dx}, \tag{3.13}$$

and combining (3.12) and (3.13) results in

$$\frac{d^2 f}{dx^2} = (1 + (\frac{df}{dx})^2)\frac{d\psi}{dx} \Leftrightarrow \frac{d\psi}{dx} = \frac{\frac{d^2 f}{dx^2}}{1 + (\frac{df}{dx})^2} = \frac{f''(x)}{1 + f'(x)^2}. \tag{3.14}$$

In conclusion (3.9), (3.11) and (3.14) gives

$$\kappa = \frac{f''(x)}{(1 + f'(x)^2)^{\frac{3}{2}}}. \tag{3.15}$$

Differentiation of (3.1) results in

$$f'(x) = K + 3Ax^2 + 2Bx,$$
$$f''(x) = 6Ax + 2B,$$

which for every sample, at the position of the preceding vehicle, gives an artificial road curvature

$$\kappa_{artificial} = \frac{6AD_x + 2B}{(1 + (K + 3AD_x{}^2 + 2BD_x)^2)^{\frac{3}{2}}}. \tag{3.16}$$

3.2.3 Other signal transformations

Because lane changes in both directions were treated the same way, no concern was given whether lateral motion was happening to the right or to the left. When it is said that D_y or $V_{y,rel}$ was used as input, it is therefore the corresponding absolute velocities that were actually used. In order to keep the covariate vector small, a signal corresponding to the speed of the preceding vehicle was also assembled,

$$v_{x,prec} = v_{x,host} + v_{x,rel}. \tag{3.17}$$

3.3 Sliding Window of Data

At any given time instant, i, it seems plausible that the inferred driver intention must be based both on current and old values of the observed variables. Therefore the effective covariate vector, $\mathbf{x}_i$, becomes

$$\mathbf{x}_i = \begin{bmatrix} x_i^1 & x_{i-1}^1 & \dots & x_{i-N_p^1+1}^1 & \dots & x_i^{N_c} & x_{i-1}^{N_c} & \dots & x_{i-N_p^c+1}^{N_c} \end{bmatrix}, \tag{3.18}$$

where N_p^j is the number of past values used for covariate j. It should be emphasized that this decision was inspired by previous works, like [28] [7], but also based on intuition. At first, N_p was selected such that the covariate vector represented a 1 second long sliding window of data, i.e $N_p = \frac{1}{T_s}$. For a short sampling time, however, this resulted in a very long input vector that made both training and prediction quite slow for multiple algorithms. Instead of reducing the length of the sliding window a down sampled number of values from the past second were used, i.e. the covariate vector was

$$\mathbf{x}_i = \begin{bmatrix} x_i^1 & x_{i-\tau_d}^1 & x_{i-2\tau_d}^1 & \dots & x_{i-N_p^1+1}^1 & \dots & x_i^{N_c} & x_{i-\tau_d}^{N_c} & \dots & x_{i-N_p^c+1}^{N_c} \end{bmatrix}, \tag{3.19}$$

where τ_d is a down sampling factor. Downsampling alone causes high-frequency signal components to be misinterpreted by subsequent users of the data, which is a form of distortion called aliasing [19]. The effect is that different signals becomes indistinguishable when sampled. By having all the signals have their high-frequency components reduced by a filter, the aliasing effect is reduced. This further justifies the lowpass filtering described in Section 3.1.3.

The Nyquist frequency is known as the folding frequency of a sampling system. It is defined as half the sampling rate. In order to avoid folding the sampled signal should not have any components above the Nyquist frequency [19]. As previously mentioned, the original sampling frequency was $f_s = 100$ Hz and the lowpass filter had a cutoff frequency of 6 Hz. Because of this, the downsampling factor was chosen as $\tau_d = 8$, which resulted in a downsampled signal with frequency $f_d = \frac{100}{8} = 12.5$ Hz. This means that the signals, even after the downsampling, did not contain any significant frequency components above half the new sampling frequency,

$$f_{Nyquist} = \frac{f_d}{2} = \frac{12.5}{2} \geq f_c = 6 \text{ Hz}. \tag{3.20}$$

4

Results and Discussion

This chapter presents the outcome of classifications for the algorithms that have been realized.

A naive Bayes classifier has been implemented alongside attempts with neural networks, support vector machines and Gaussian processes. As training data 55 occasions of vehicle following that ended with a lane change were used. Each of these occasions, where there were a preceding vehicle present, lasted for about 60 seconds in average and the whole training set consists of approximately 55 minutes of driving. The training data was labelled as described in Section 3.1, with each lane switch being assumed to be five seconds long. This means that about 96 percent of the training data was made up by passive driving.

A systematic way of presenting the results and comparing the different classifiers was sought. In order to achieve this it was necessary to limit the analysis of the advanced classifiers, to a well performing subset of sensor signals that yielded high accuracy when presented to the benchmark model, naive Bayes.

A couple of different metrics to compare models were also considered. Two thirds of the data were used for training and the rest was held out for validation. Testing was done using 5-fold-cross-validation on the two thirds used for training. All plots in the following sections are results from presenting the same third of validation data to different algorithms. The validation data consists of 21 lane switches in total. In order to grant the reader a comprehensive view of the classifiers capability, each classifier is presented with a figure including both the true output and the predicted output. In these figures the true output is marked as a solid orange line and the predicted class at each time step is marked with a blue star. The state for lane change is labelled as "1" and regular driving is labelled as "0". The performance metrics in Section 2.3 do also provide easily accessible measurements for model comparison. The meaning of a positive output is a lane switch. For the sake of a systematic model comparison, all the classifiers that

provided a probabilistic output had their output discretized as,

$$\begin{cases} \hat{y}_i = 1 & \text{if } Pr(Y = 1|\mathbf{x}_i) \geq 0.5. \\ \hat{y}_i = 0 & \text{if } Pr(Y = 1|\mathbf{x}_i) < 0.5. \end{cases} \tag{4.1}$$

In order to keep the report compact the confusion matrix, in its original matrix form, is excluded. The FPR, TPR, $precision$ and $accuracy$ are instead provided in plain text and are summarized in Table 4.1, which is presented in Section 4.6.

Something that should be emphasized about the performance metrics are that they were computed sample to sample. This means that for each lane switch of 2.5 seconds there were $\frac{2.5}{T_s} = \frac{2.5}{0.01} = 250$ time stamps that should ideally be predicted as a lane switch. Another idea was to declare a classified lane switch as every occurrence of a set of consecutive predictions of a predetermined size. For example 20 samples in a row, with a predicted output labeled as lane switch, would confirm an actual switch. For model comparison purposes it was decided that sample to sample metrics were sufficient. It should however be pointed out that a low rate of true positive lane switches, does not necessarily mean that a lot of lane switches were missed. Instead it can mean that many of the lane switches were not predicted all the 2.5 seconds in advance, but just at the very end. Should the system be implemented in a truck and be used in real-time it is probable that a more sophisticated threshold, for when to alert cooperating systems, would have to be developed.

4.1 Naive Bayes

It was decided to use naive Bayes as a first implementation and later use it as a reference due to the fact that both training of models and prediction is computationally done very fast.

To begin with the transformed signal signifying "minimum distance to the lane edge", Δ_{lane}, was used. This signal requires road markers. The result was fair which demonstrated some potential in naive Bayes as a classifier. The result is presented in Figure 4.1.

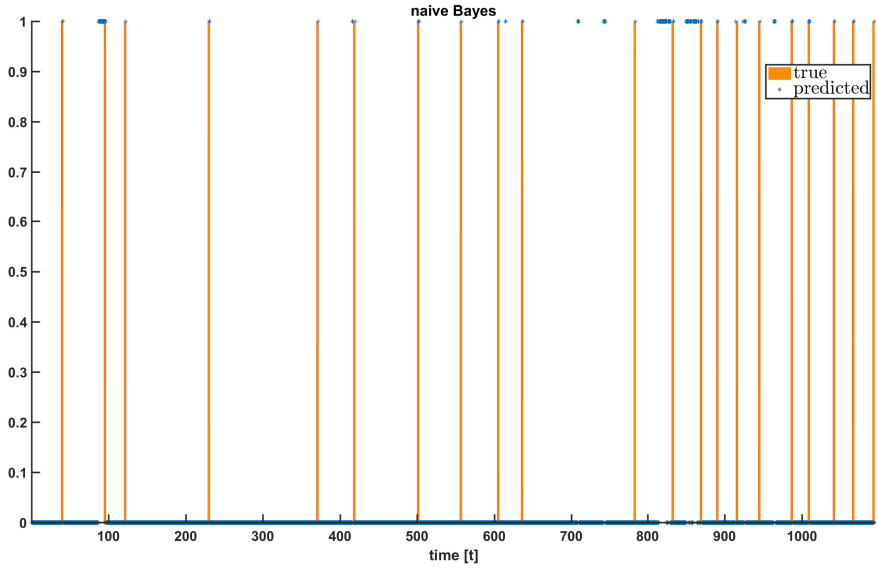

Figure 4.1: *The naive Bayes classifier utilizing the signal Δ_{lane} that requires visible road markers. TPR = 26.1 %, FPR = 4.2 %, precision = 24.1 %, accuracy = 92.5 %.*

When the road marker signal was removed and solely vehicle motion and distances were provided as input the classifier still did quite well. The result in Figure 4.2 is unsatisfactory primarily because of the extensive misclassifications, about 800 seconds into the validation data. It is due to these false positives that the precision is very low. The troublesome lane switches were an issue for the classifier utilizing Δ_{lane} as well and hence they were studied more thoroughly. When watching the logged video stream it was discovered that the road in these occasions had a large curvature. It was in order to successfully be able to predict lane switches on curvy roads as well, that the idea to utilize the road curvature, to further enhance the input vector, materialized.

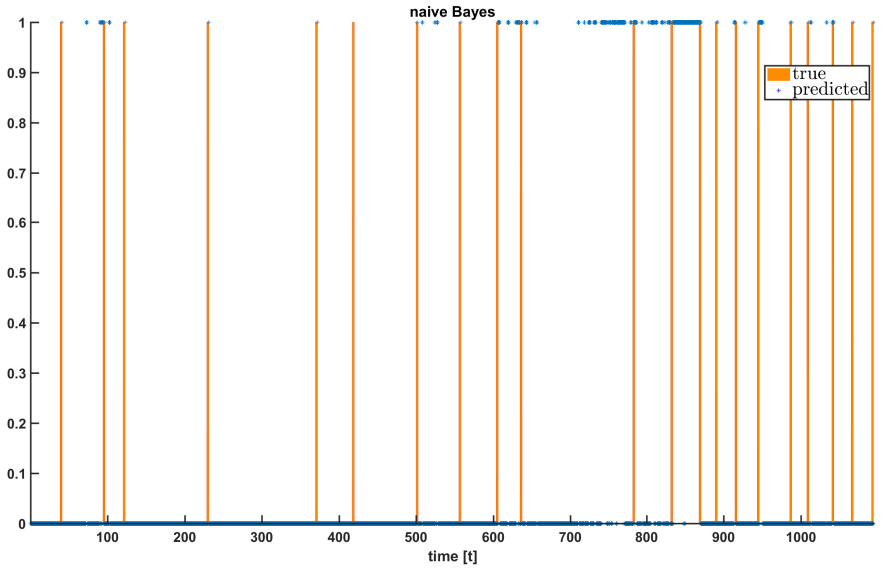

Figure 4.2: *The naive Bayes classifier solely using the signals D_x, D_y, $v_{x,prec}$ and $v_{y,rel}$. TPR = 25.8 %, FPR = 10.9 %, precision = 10.7 %, accuracy = 86.0 %.*

Utilizing the signal deviation from expected lateral velocity, ΔE_{v_y}, as sole input did, as expected, not result in a tolerable outcome. When used alongside some other signals like speed of the preceding vehicle and the absolute value of its lateral motion it did however result in a promising classifier. The result is given in Figure 4.3. It was overall not that much better than the classifier presented in Figure 4.2, but the number of misclassifications were reduced, which shows in the reduced FPR and improved precision.

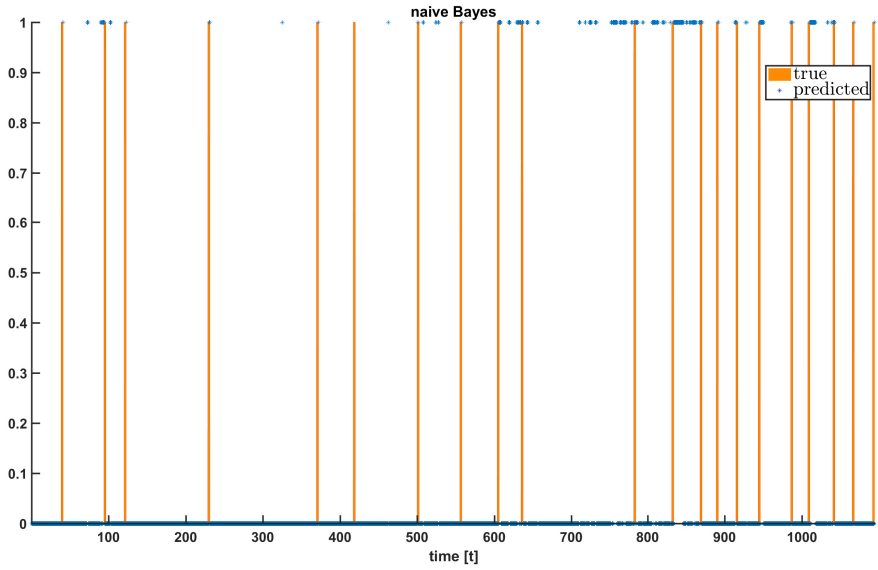

Figure 4.3: *The naive Bayes classifier using the signals:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}. *TPR* = 26.5 %, *FPR* = 8.2 %, *precision* = 14.0 %, *accuracy* = 88.6 %.

Up until this point a multitude of sets of sensor signals were tested but the sensor set, $\{D_y, v_{x,prec}, v_{y,rel}, \Delta E_{v_y}\}$ provided the best performance when using naive Bayes. It was therefore decided to concentrate the forthcoming analysis on using that specific set. The cutoff frequency of 6 Hz, used by the lowpass filter described in Section 3.1.3, was also settled upon. Using old signal values from the past second seemed to be enough to capture the sought after driving pattern because a lower number of old values gave worse results and a higher number did not improve the results further. Instead it solely increased the computational time, both for training and prediction.

4.2 Artificial Neural Network

Tests were also carried out using neural networks. A couple of different activation functions were used for testing, but for the neurons in the output layer the logistic activation (2.16) was used consistently. This was done in order to get a valid probabilistic interpretation of the output. The learning rate, α, was fixed to 0.01. As described in Section 2.2.2 the structure of the ANN is decided by the number of hidden layers and the number of neurons in each of these. The goal is to create a hidden structure that is neither too complex nor too simple. A neural network using a too complex hidden structure, will take a long time to train and there is also a high risk that a local minimum is encountered during training. On the other hand, a too simple hidden structure might not be able to solve the given problem. A good starting point is a single hidden layer, with a number of neurons equal to twice the size of the input layer. Depending on the performance of that network, the number of neurons in the hidden layer can either be increased or decreased [16].

Training networks, by following this rule-of-thumb, resulted in a best performing network of 16 neurons. This is when using an input-vector of 48 elements: the 4 different signals $\{D_y, v_{x,prec}, v_{y,rel}, \Delta E_{v_y}\}$ and 12 past values of each. A graph showing the result is provided in Figure 4.4. The low rate of false positives is appealing but the low TPR is a set back.

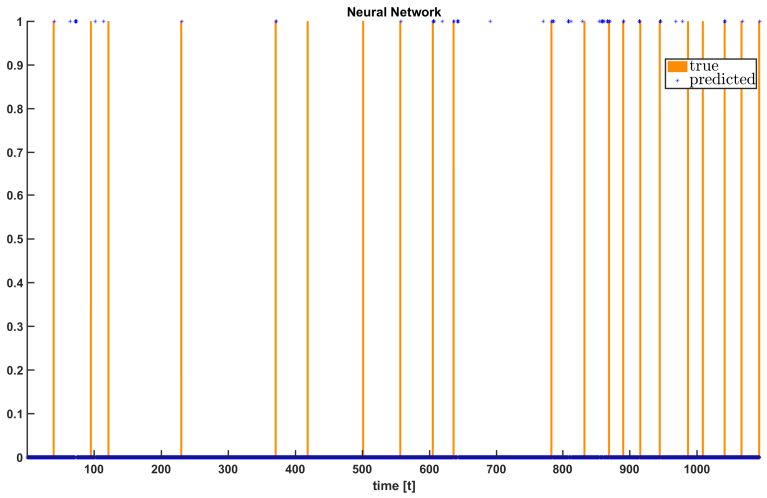

Figure 4.4: *Artificial neural network using one hidden layer consisting of 16 neurons. The activation function used was the hyperbolic tangent function, (2.17). The signals used were: D_y, $v_{x,prec}$, $v_{y,rel}$ and ΔE_{v_y}. TPR = 15.7 %, FPR = 1.57 %, precision = 33.6 %, accuracy = 94.4 %.*

Some research has indicated that a second hidden layer is rarely of any value [16]. A couple of tests were nonetheless carried out using a second hidden layer

and the performance of the network was improved a bit, see Figure 4.5. The performance did not increase further by adding a third layer, and did not improve with a higher amount of neurons than 2 in the second layer either. A structure of two hidden layers with 16 neurons in the first and 2 neurons in the second was therefore settled upon. The structure of two hidden layers, with 16 and 2 neurons in each respectively, is by no means confirmed to be optimal, but among the tested ones it provided the best results.

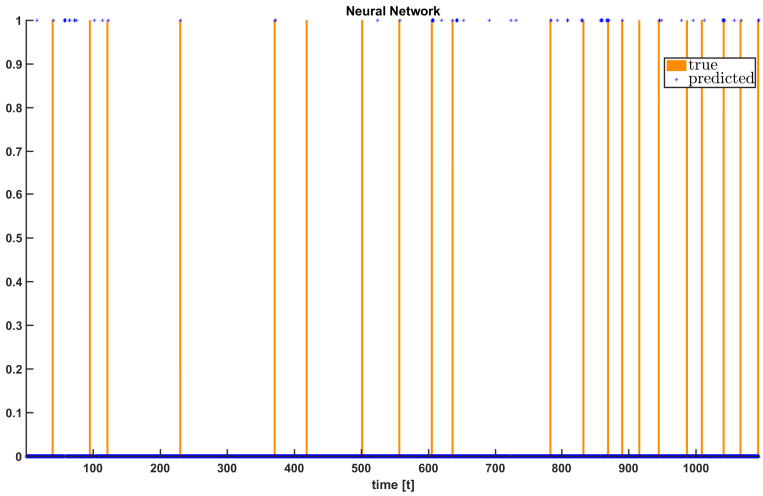

Figure 4.5: *Artificial neural network using two hidden layers with 16 and 2 neurons in each. The activation function used was the hyperbolic tangent function,* (2.17)*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*.* TPR = 14.8 %, FPR = 1.0 %, *precision* = 42.6 %, *accuracy* = 94.9 %.

As mentioned in Section 2.2.2 each link in the network has a weight, $w_{i,j,l}$, associated with it. The neural network used in Figure 4.4 had a structure of one hidden layer with 16 neurons. This together with the input layer, which consisted of 48 elements, and an output layer with one neuron resulted in $48 \cdot 16 + 16 \cdot 1 + 16 + 1 = 801$ weights that were estimated during training. The neural network used in Figure 4.5 had a structure of two hidden layers with 16 and 2 neurons in each respectively. This together with the same input layer, of 48 elements, resulted in $48 \cdot 16 + 16 \cdot 2 + 2 \cdot 1 + 16 + 2 + 1 = 821$ weights that were estimated during training. Because the number of weights is multiplicative, and the dominant term in both equations is $48 \cdot 16$, adding a second layer with only 2 neurons did not increase the number of weights considerably. The computational complexity, of on-line usage of these two networks, are therefore about the same.

Different activation functions were also tested. Training with the back propagation algorithm does not work when using the hard threshold, given by (2.15), because there is no derivative in the origin. Using a hard threshold as activation with a different learning algorithm resulted in a passive classifier that always

predicted the negative class. Since the negative class makes up 96 percent of the training data, a classifier like that has higher accuracy, than a classifier that often predicts false positives. This might explain why a passive behaviour was favoured during training. Attempts were also made using the logistics function, (2.16). The result is provided in Figure 4.6.

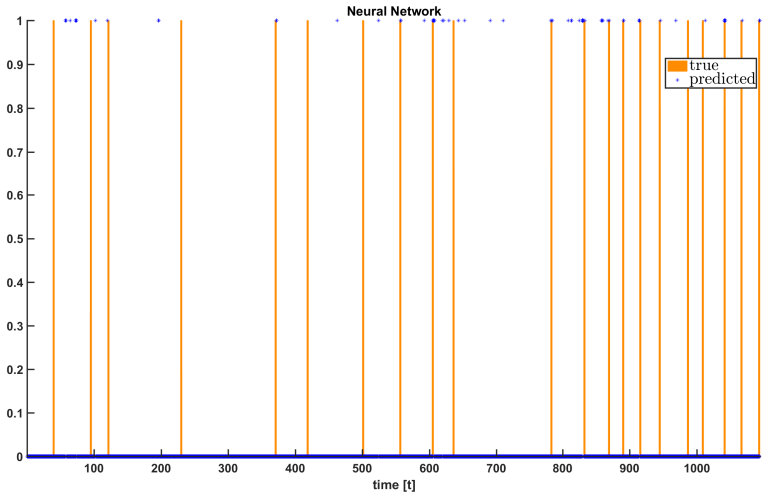

Figure 4.6: *Artificial neural network using two hidden layers with 16 and 2 neurons in each. The activation function used was the logistic function,* (2.16)*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*.* $TPR = 17.9$ %*,* $FPR = 1.68$ %*, precision* $= 34.9$ %*, accuracy* $= 94.4$ %*.*

The results received when using the logistic activation function and the hyperbolic tangent function are very similar. The rate of true positives is slightly higher for the network using a logistic activation but at a cost of having a higher rate of false positives as well as a lower precision. The overall accuracy is about the same. Using any of these two activation functions, it should probably be possible to obtain the same result. The reason that the results are a bit different is, likely, due to the fact that different local optima were encountered during training.

4.3 Support Vector Machines

Tests were also carried out using support vector machines. The three kernel functions, (2.38), (2.39) and (2.40) were used. For the polynomial kernel both a quadratic version, $\gamma = 2$, and a cubic version, $\gamma = 3$. SVM with a linear kernel did, as the neural network using a hard threshold as activation, result in a passive classifier that always predicted the negative class. Likely for the very same reason. Using a quadratic or cubic kernel, Figure 4.7 and 4.8, did also give underwhelming results. Classifying with a quadratic kernel resulted in a very low rate of true positives. The classifier with a cubic kernel did classify almost all the lane switches correctly, but the rate of false negatives was high. Usage of the Gaussian kernel function did, on the other hand, result in a very promising classifier. That classifier is presented in Figure 4.9. For the classifier using a Gaussian kernel, $\gamma = 0.021$, was decided by cross validation.

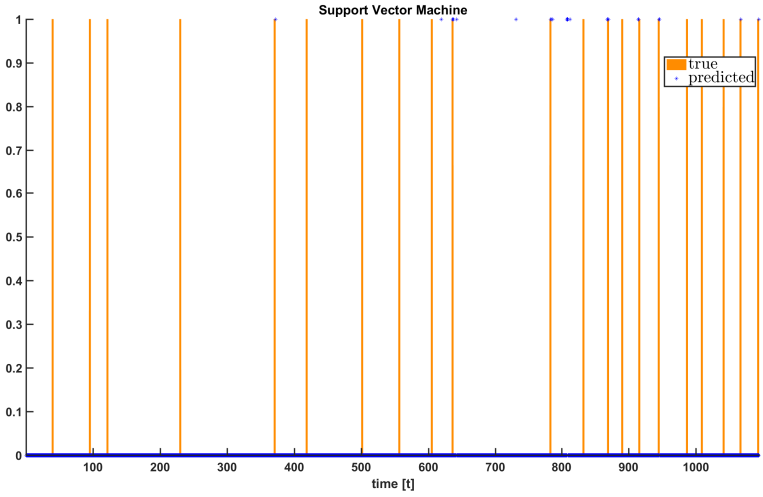

Figure 4.7: *Support vector machine using a quadratic kernel function,* (2.39) *with* $\gamma = 2$*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*.* $TPR = 4.84\,\%$*,* $FPR = 0.27\,\%$*, precision* $= 48.0\,\%$*, accuracy* $= 95.2\,\%$*.*

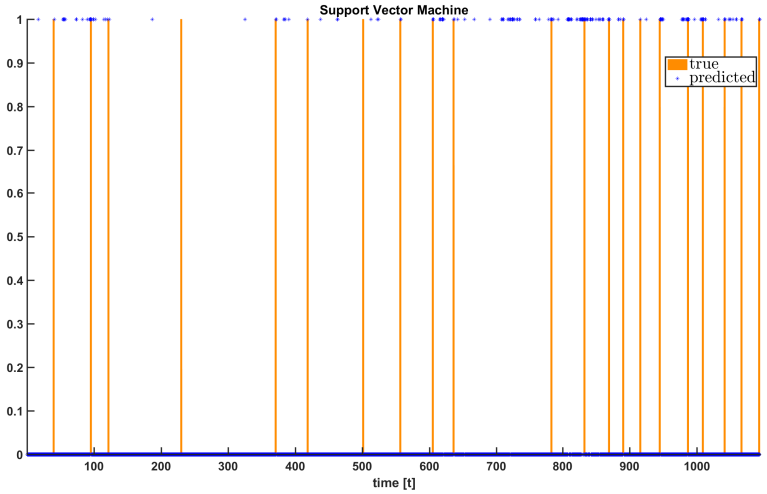

Figure 4.8: *Support vector machine using a cubic kernel function,* (2.39) *with* $\gamma = 3$*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*. TPR* = 23.8 %*, FPR* = 2.63 %*, precision* = 31.4 %*, accuracy* = 93.8 %*.*

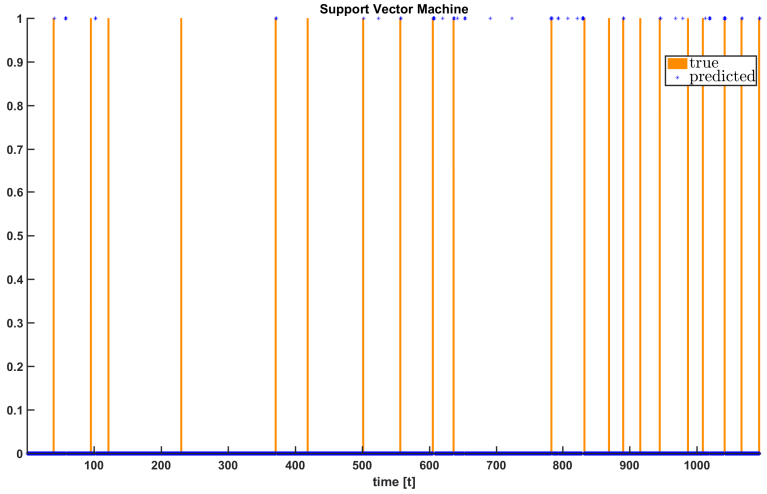

Figure 4.9: *Support vector machine using a Gaussian kernel function,* (2.40)*,* $\gamma = 0.021$*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*. TPR* = 15.0 %*, FPR* = 0.86 %*, precision* = 47.0 %*, accuracy* = 95.1 %*.*

4.4 Gaussian Processes

For classification with Gaussian processes the same three kernel functions, (2.38), (2.39) and (2.40) were used, as for SVMs. For the polynomial kernel both a quadratic version, $\gamma = 2$, and a cubic version, $\gamma = 3$. The result from using a linear kernel, Figure 4.10, was poor. Using a quadratic kernel, Figure 4.11, gave decent results, but quite a few of the lane switches in the validation data were missed all together. The classifier with a cubic kernel, Figure 4.12, did worse than the one using a quadratic kernel. Usage of the Gaussian kernel function resulted in the best performing classifier, primarily because of a much higher rate of true positives. The classifier using a Gaussian kernel is presented in Figure 4.13. For that classifier, γ, was decided by cross validation.

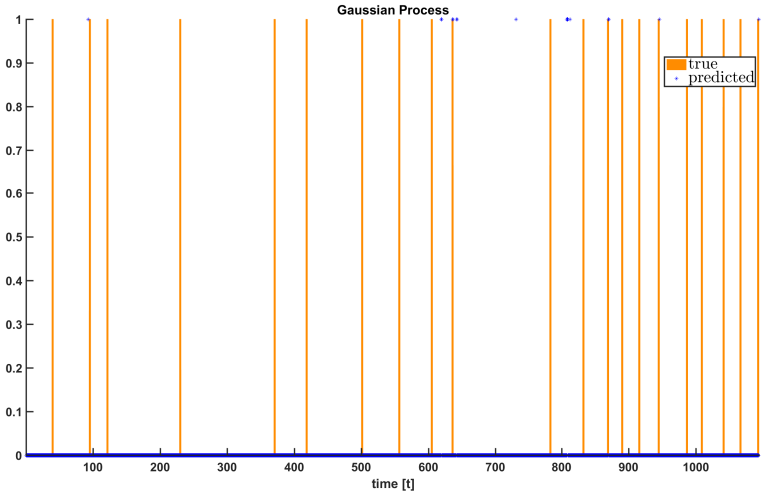

Figure 4.10: *Classification with a Gaussian process, using the Laplace approximation method and a linear kernel function,* (2.38)*. The signals used were:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}*.* $TPR = 3.85$ %, $FPR = 0.34$ %, *precision* $= 36.6$ %, *accuracy* $= 95.0$ %.

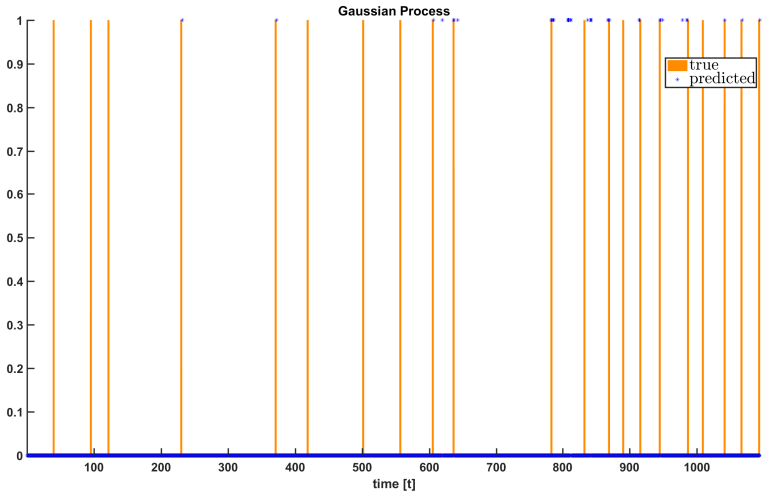

Figure 4.11: *Classification with a Gaussian process, using the Laplace approximation method and a quadratic kernel function, (2.39) with $\gamma = 2$. The signals used were: D_y, $v_{x,prec}$, $v_{y,rel}$ and ΔE_{v_y}. TPR = 7.01 %, FPR = 0.45 %, precision = 44.2 %, accuracy = 95.1 %.*

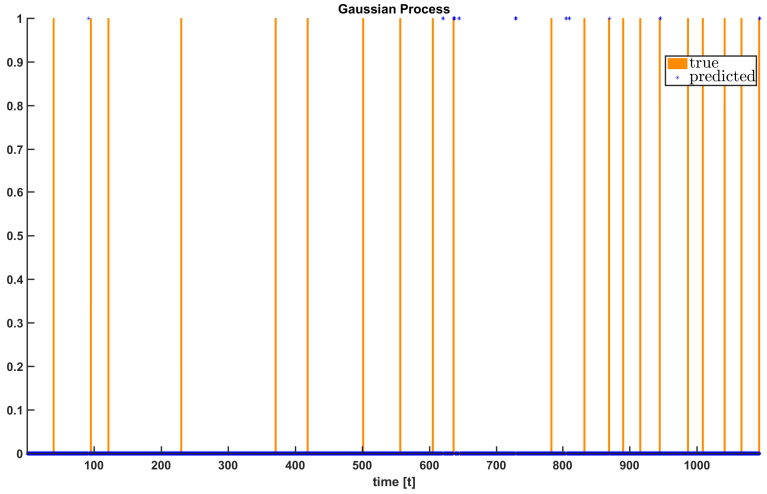

Figure 4.12: *Classification with a Gaussian process, using the Laplace approximation method and a cubic kernel function, (2.39) with $\gamma = 3$. The signals used were: D_y, $v_{x,prec}$, $v_{y,rel}$ and ΔE_{v_y}. TPR = 3.83 %, FPR = 0.34 %, precision = 36.5 %, accuracy = 95.0 %.*

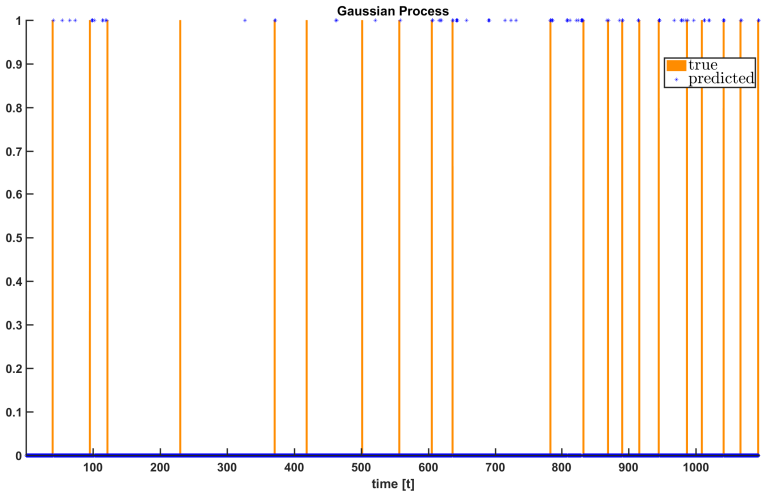

Figure 4.13: *Classification with a Gaussian process, using the Laplace approximation method and a gaussian kernel function,* (2.40)*. The signals used were:* D_y*,* $v_{x,prec}$*,* $v_{y,rel}$ *and* ΔE_{v_y}*.* $TPR = 13.2$ %*,* $FPR = 0.93$ %*, precision* $= 41.8$ %*, accuracy* $= 94.9$ %*.*

4.5 Multiclass Classification

For the system to be useful it was required to lower the rate of false positives even further, especially on curvy roads. A theory was that lane switches to the right would, more often, be characterized by a braking pattern because most slip roads, on highways, are to the right. Lane switches to the left would on the contrary be characterized by an accelerating pattern because of them being parts of the initiation of overtakings. To test this theory a histogram was plotted, showing the distribution of the 55 lane switches used in the training data. The result is in Figure 4.14. It can clearly be seen that the theory is valid.

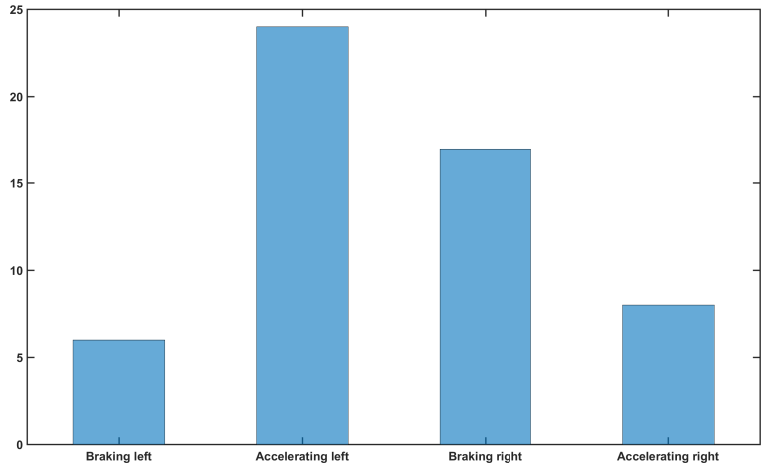

Figure 4.14: *A histogram partitioning the number of cases where a lane switch was done with an increase of speed and when it was done in a braking manner. This partitioning was done for lane switches to the left and right respectively.*

In order to improve the performances of the trained classifiers it was therefore decided to try multiclass classification, separating the two distinguishable cases with lane switches to the right and left, as two different classes. The output was labelled as "+1" for lane switches to the left, "−1" for lane switches to the right and regular driving was still labelled as "0". For this further study, only the two most promising techniques were evaluated. The SVM, ANN and GP classifiers had all got decent performance results. The computational complexity for prediction, when using GPs, was however considered to be too much of a set back. The two, so far, best performing classifiers, using neural networks, were the ones presented in Figure 4.5 and 4.6. Among these two, it was decided to carry on using the one with a hyperbolic tangent as activation function, due to the lower FPR.

As mentioned in Section 2.4 the usage of neural networks provides a straightforward extension to the multiclass problem. Instead of having one neuron in the

output layer, three neurons with probabilistic output were used. Each of these neurons output, $\{y_{-1}, y_0, y_{+1}\}$, represented the probability that a set of inputs belonged to one of the three classes. The predicted class was then chosen as

$$\hat{y} = \underset{c\in\{-1,0,+1\}}{\text{argmax}}\ y_c(\mathbf{x}). \tag{4.2}$$

It can be seen in Figure 4.15 that some of the lane switches to the right are misclassified as lane switches to the left and vice-versa. This is however not a problem because the only system demand is to separate lane switches from regular driving. In order to be able to compare the result of this classifier with the ones previously attained, all occurrences of lane switches to the right were relabelled as +1. This was done after the initial classification was completed. The result is presented in Figure 4.16.

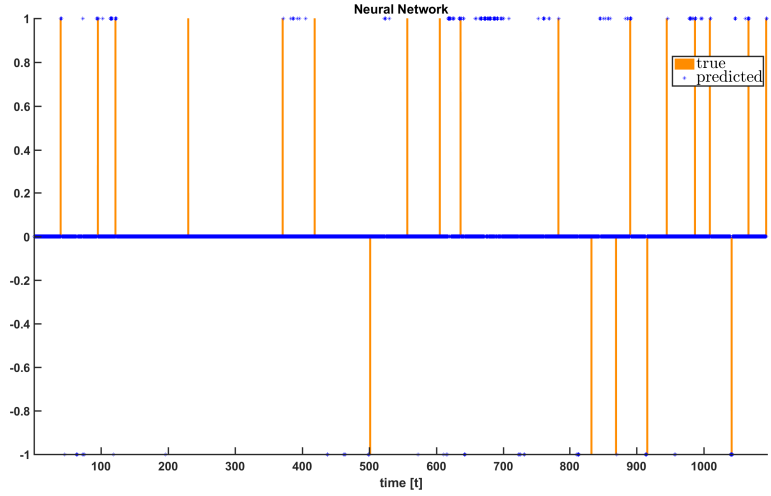

Figure 4.15: *Artificial neural network using two hidden layers with 16 and 2 neurons in each and an output layer consisting of 3 neurons. Lane switches to the left are labelled as "+1" and lane switches to the right are labelled as "-1". 0 means regular driving. The activation function used was the hyperbolic tangent function, (2.17). The signals used were:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}.

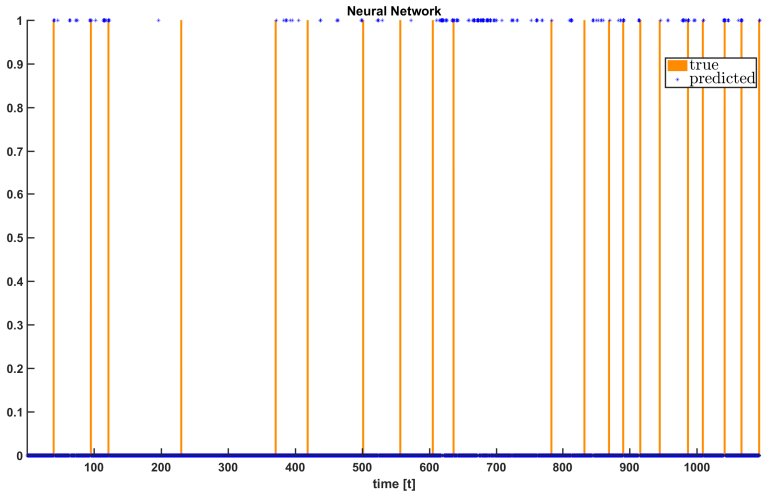

Figure 4.16: *Artificial neural network using two hidden layers with 16 and 2 neurons in each and an output layer consisting of 3 neurons. Both lane switches to the left and right are labelled as "+1" and regular driving is labelled as "0". The activation function used was the hyperbolic tangent function,* (2.17)*. The signals used were:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}. *TPR* = 22.3 %, *FPR* = 3.61 %, *precision* = 23.8 %, *accuracy* = 92.8 %.

For SVM the one-versus-one strategy, was used. Predictions in the ambiguous region was labelled as regular driving. As for the case of the ANN, all occurrences of lane switches to the right were relabelled as +1 after the initial classification was completed. In Figure 4.17 it can be seen that none of the lane switches to the right are correctly classified. This indicates that having an extra class label is redundant. The result after the relabelling, in Figure 4.18, shows that the classifier, however got the highest overall accuracy of all the presented ones.

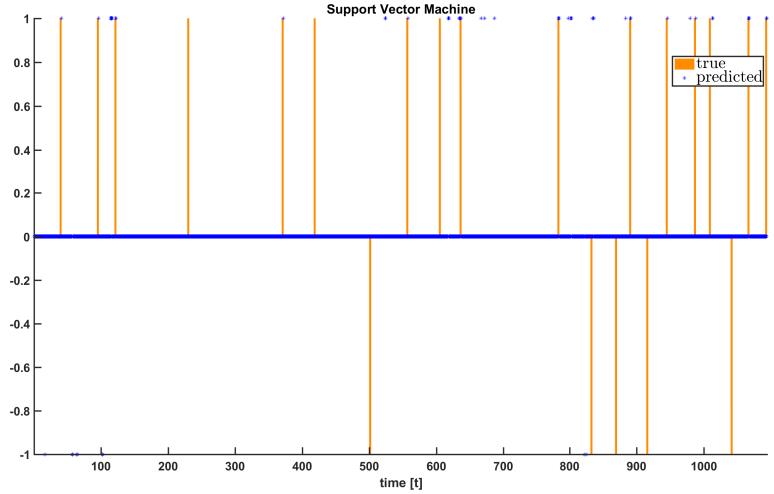

Figure 4.17: *A classifier utilizing 3 support vector machines with Gaussian kernel functions, (2.40), and the one-versus-one strategy. Lane switches to the left are labelled as "+1" and lane switches to the right are labelled as "-1". 0 means regular driving. The signals used were:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}.

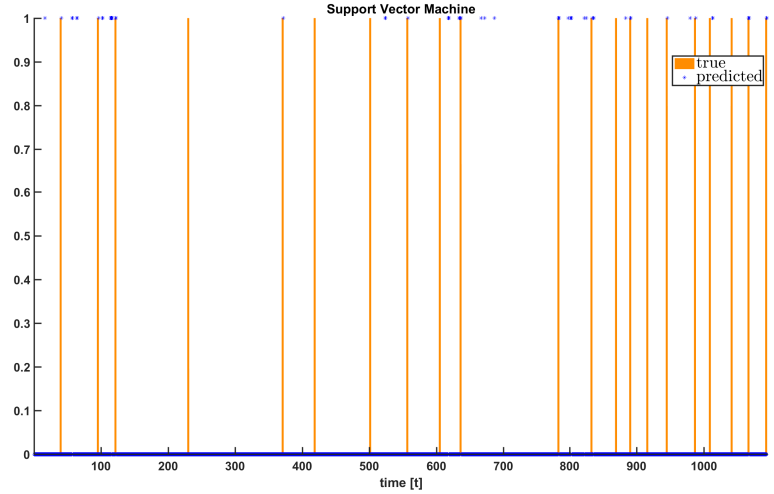

Figure 4.18: *A classifier utilizing 3 support vector machines with Gaussian kernel functions, (2.40), and the one-versus-one strategy. Both lane switches to the left and right are labelled as "+1" and regular driving is labelled as "0". The signals used were:* D_y, $v_{x,prec}$, $v_{y,rel}$ *and* ΔE_{v_y}. *TPR* = 15.4 %, *FPR* = 0.90 %, *precision* = 39.8 %, *accuracy* = 96.0 %.

4.6 Conclusion

All the best performing classifiers, one for each of the studied techniques, are summarized in Table 4.1. A ROC graph for these classifiers is provided in Figure 4.19. The naive Bayes classifier, first presented in Figure 4.3, and the neural network using multiple neurons in the output layer, presented in Figure 4.16, have got the highest rates of true positives. They do on the other hand have the highest rates of false positives as well. The outcome of having the algorithm output an upcoming lane switch, of the preceding vehicle, would eventually be to alert the driver that manual steering is required. Having low false alarm rates is essential for all human-machine interfaces like this, because of the level of annoyance generated by an incorrect analysis.

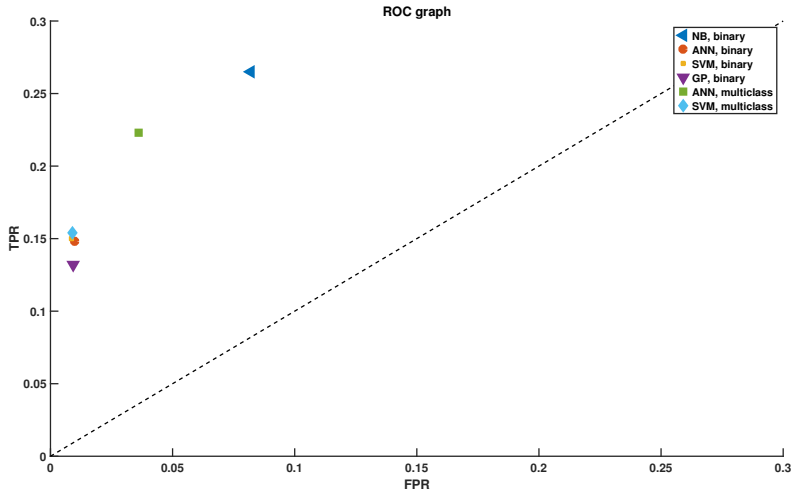

Figure 4.19: *A ROC graph, for the best performing classifiers, one for each of the studied techniques.*

	Type	*TPR*	*FPR*	*precision*	*accuracy*
NB	Binary	26.5	8.2	14.0	88.6
ANN	Binary	14.8	1.0	42.6	94.9
SVM	Binary	15.0	0.86	47.0	95.1
GP	Binary	13.2	0.93	41.8	94.9
ANN	Multiclass	22.3	3.61	23.8	92.8
SVM	Multiclass	15.4	0.90	39.8	96.0

Table 4.1: *The best performing classifiers, one for each of the studied techniques.*

The binary SVM, ANN and GP classifiers have all got comparable performance results. The basic complexity for the prediction process, when using GPs, of $O(N^3)$ makes it almost intractable to implement in the processors that trucks, according today's standards, are equipped with.

The algorithm developed in this thesis is supposed to operate in a wide network of system functions. Should the system be implemented in a truck it is probable that a sophisticated threshold, for when to actually alert cooperating systems of an upcoming lane switch, would have to be developed. For this task it would be useful to have a level of certainty related to the classifiers output. A major set back with the support vector machine is its incapability to provide probabilistic output. This is something that has not been reflected by the presented performance metrics.

For all the evaluated classifiers it is required to store all the signals, that are being used as covariates, from the past second. This means keeping $\frac{N_{cov}}{T_s}$ signal values in memory. The ANN with binary output, introduced in Figure 4.5, used 821 weights. The complexity of prediction using this network is not insignificant and consists of roughly 800 computations of the activation function and a number of summations. It is also required to store these 821 values and together with the stored signal values, the memory requirement is quite high.

Previous work in the area and the results obtained in this thesis, do further conclude that prediction of lane-changes is a nontrivial task. This is mainly because the inference of driver intents has a natural uncertainty to it. Even if a noise-free measurement of the input was available, the driver's intention would not be deterministic anyway. In order to make a qualified guess of the drivers intention, it is required to study cues from both the vehicle itself and the environment. Having a low rate of false positives is essential. The action of including a signal that utilizes the road curvature did help in improving this rate. Another action taken in order to further reduce the FPR was to separate lane switches to either side as two different classes. The histogram, presented in Figure 4.14, did indicate that a classifier with a binary, linear decision rule, might struggle. The results in Table 4.1 and Figure 4.19 do, on the other hand, show better results for the binary classifiers. A more thorough investigation, of the usage of multiple classes, is however required in order to conclude anything of certainty.

Apart from the lack of probabilistic output, the SVM with a Gaussian kernel function, is the best performing classifier. The rates of true positives of that classifier, and most of the others as well, are quite low. A limitation with supervised learning is that the model can never be better than the supervisor, who labelled the training data. Labelling the output data was a cumbersome process and an assumption had to be made, that each lane switch was 5 seconds long. This assumption will not be true for all the lane switches, which makes it essentially impossible, for a classifier, to excel with perfection. A TPR higher than 15 percent could, however, be expected.

It is likely that more fine tuning of design parameters, or a more sophisticated kernel function, would yield improved performance results. By taking a rough look at Figure 4.9, it can be seen that the classifier, at least for the most parts, has a higher density of lane switch predictions around actual lane switches. If that

information could be utilized, it would be possible to predict up to 14 of the 21 lane switches in the validation data and call for somewhere between 9 and 13 false positives in the process. This would depend on the thresholding that was discussed in the beginning of this chapter.

Some of the sensors required for tracking might lose performance in poor light conditions or certain weather types, like fog and snow. As mentioned in Section 1.3 and 3.1, the thesis was therefore delimited to only include cases when the tracking signals were available with moderate frequency. If the tracking was not to be working the results would therefore, probably, be much worse. For example the signals originating from the camera sensor might be less accurate during the night or in heavy snowfall. It is likely that the proposed algorithm in those cases would not be useful at all.

4.7 Future Work

A multitude of sensor data is available through already existing sensor fusion and the original signals can always be combined in new ways, something that can make it possible to more compactly present input to the classifier. This study of finding useful signals can always be extended. Lane switches done by heavy duty vehicles do probably differ from lane switches done by regular cars. It might be of interest to take information about the type of the preceding vehicle into account, in the covariate vector.

With more advanced map data, multiple interesting covariates could also have been extracted. [7] had their classifier output a turn maneuver, only within 30 meters of a crossing, based on digital maps. For this thesis it might have been helpful to limit the classifier to output a highway drop off, only within a given distance of a slip road. Similarly, a lane switch should only be classified as positive, if there exists an actual lane for the preceding vehicle to steer into. [20] proposed a way to estimate the number of available traffic lanes on a given GPS position. Utilizing a signal like this could potentially reduce the rate of false positives further. In order to not have the classifier be tricked by an accelerating pattern caused by something other than a lane switch, e.g. a hilly terrain, it might also be useful to include a height profile for the current road segment.

A methodology that was developed by Michael E. Tipping, [25], in order to address the problem with SVMs only making point predictions, rather than predictive distributions, is the relevance vector machine, RVM. The method can be viewed as a Bayesian extension of the SVM framework. Apart from the probabilistic output, another benefit of using RVM instead of SVM is that it is not necessary to estimate the margin trade-off parameter, C. This is an estimation process that otherwise depends upon a cross-validation procedure, which increases the amount of required training data. A disadvantage of RVM, compared to SVM, is a high complexity of the training phase. For the training phase it is required to perform $O(N^3)$ computations which makes training considerably slower than in the case of support vectors [25]. This would be a minor problem because the learning phase takes place off-line. Prediction is done with complexity equal to that of SVM because the RVM and the SVM have identical functional form [25].

For this thesis 55 lane switches, corresponding to 328.000 samples of training data was used. SVMs are nonparametric models and, with a structure as complex as two hidden layers and 821 weights, the ANN does require a lot of data. For these two methods it is likely that the results would be improved by having a higher amount of example input-output pairs. Bayesian approaches bring together work in the machine learning and the statistics community and after having studied related works, when writing this report, it is clear that their accessibility and popularity continues to grow. For the time being, the computational complexity is too much of a set back for it to be feasible to implement any of them in a real truck, but Bayesian methods are generally strongest when the amount of training data is limited. For this thesis several thousand kilometers of driving data was available but that might not be the case for all tasks within the vehicle industry, where machine learning is required.

Bibliography

[1] Christopher Urmson, Joshua Anhalt. Autonomous driving in urban environments: Boss and the Urban Challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge*, pages 425–466, 2008. doi: 10.1002/rob.20255. Cited on page 3.

[2] Sylvain Arlot. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2009. doi: 10.1214/09-SS054. Cited on page 34.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN 0-387-31073-8. Cited on pages 2, 5, 11, 14, and 29.

[4] Reid Simmons, Brett Browning et al. Learning to Predict Driver Route and Destination Intent. *Intelligent Transportation Systems Conference*, pages 127 – 132, 2006. doi: 10.1109/ITSC.2006.1706730. Cited on pages 3 and 5.

[5] Christopher J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. doi: 10.1023/A:1009715923555. Cited on page 22.

[6] Wen Yao, Huijing Zhao, Franck Davoine et al. Learning lane change trajectories from on-road driving data. *Intelligent Vehicles Symposium*, IV:885 – 890, 2012. doi: 10.1109/IVS.2012.6232190. Cited on pages 3 and 12.

[7] Holger Berndt, Jorg Emmert, Klaus Dietmayer. Continuous Driver Intention Recognition with Hidden Markov Models. *Intelligent Transportation Systems*, pages 1189 – 1194, 2008. doi: 10.1109/ITSC.2008.4732630. Cited on pages 2, 3, 5, 43, and 65.

[8] E. Donges. A Two-Level Model of Driver Steering Behavior. *Human Factors*, 20(6):691–707, 1978. doi: 10.1177/001872087802000607. Cited on page 3.

[9] R. Sayed, A. Eskandarian. Unobtrusive drowsiness detection by neural network learning of driver steering. *Proceedings of the Institution of Mechanical Engineers Part D Journal of Automobile Engineering*, pages 969 – 975, 2001. doi: 10.1243/0954407011528536. Cited on pages 3 and 12.

[10] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, page 861–874, 2006. doi: 10.1016/j.patrec.2005.10.010. Cited on pages 26 and 27.

[11] Robert E. Fenton. On the Steering of Automated Vehicles: Theory and Experiment. *IEEE Transactions on Automatic Control*, AC-21(3):306 – 315, 1976. doi: 10.1109/TAC.1976.1101230. Cited on page 3.

[12] Robert E. Fenton. On the Optimal Design of an Automotive Lateral Controller. *IEEE Transactions on Vehicular Technology*, 37(2):306 – 315, 1988. doi: 10.1109/25.9890. Cited on page 3.

[13] Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction*. Springer, 2008. ISBN 978-0-387-84857-0. Cited on pages 2, 5, 14, 20, 22, and 33.

[14] Roger Frigola-Alcalde. *Bayesian Time Series Learning with Gaussian Processes*. PhD thesis, University of Cambridge, 2015. Cited on pages 23 and 24.

[15] Simon O. Haykin. *Neural Networks and Learning Machines*. Pearson, 1994. ISBN 0131471392. Cited on pages 2, 5, 14, 17, and 19.

[16] Jeff Heaton. *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc, 2011. ISBN 1604390085. Cited on page 50.

[17] Meng Lu, Kees Wevers, Rob Van Der Heijden. Technical feasibility of advanced driver assistance systems (ADAS) for road traffic safety. *Transportation Planning and Technology*, 28(3):167–187, 2005. doi: 10.1080/03081060500120282. Cited on page 3.

[18] C. Sentouh, P. Chevrel, F. Mars and F. Claveau. A Sensorimotor Driver Model for Steering Control. *Systems, Man and Cybernetics*, pages 2462–2467, 2009. doi: 10.1109/ICSMC.2009.5346350. Cited on page 3.

[19] Fredrik Gustafsson, Lennart Ljung, Mille Millnert. *Signal Processing*. Studentlitteratur AB, 2011. ISBN 978-91-44-05385-1. Cited on page 43.

[20] Nobuyoshi Sato, Tsuyoshi Takayama, Yoshitoshi Murata. Estimating the Number of Lanes on Rapid Road Map Survey System Using GPS Trajectories as Collective Intelligence. *Network-Based Information Systems*, pages 82 – 88, 2012. doi: 10.1109/NBiS.2012.49. Cited on page 65.

[21] Stuart Russel, Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson, 2016. ISBN 0136042597. Cited on pages 2, 5, 8, 10, 11, 12, 13, 14, 17, 18, 19, 21, 22, 23, and 34.

[22] Erik Olsen. *Modeling slow lead vehicle lane changing*. PhD thesis, Virginia Polytechnic Institute and State University, 2003. Cited on page 2.

[23] Toru Kumagai, Yasuo Sakaguchi et al. Prediction of Driving Behavior through Probabilistic Inference. *IEICE Transactions on Information and Systems*, pages 117 – 123, 2006. doi: 10.1093/ietisy/e89-d.2.857. Cited on pages 3 and 5.

[24] T. P. Le, L. V. Dat, I. Stiharu. Application of Neural Networks to Design the Controller for Autonomous Vehicles by Learning Driver's Behavior. *Control, Automation and Information Sciences*, pages 1–6, 2014. doi: 10.1109/ICCAIS.2013.6720520. Cited on page 3.

[25] Michael E Tipping. The Relevance Vector Machine. *Advances in Neural Information Processing Systems 12*, pages 652–658, 2000. Cited on page 65.

[26] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics*. Chapman and Hall, 2002. ISBN 1584883472. Cited on page 40.

[27] Carl Edward Rasmussen, Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 10 0-262-18253-X. Cited on pages 11, 12, 23, 24, and 25.

[28] Joel McCall, David Wipf et al. Lane Change Intent Analysis Using Robust Operators and Sparse Bayesian Learning. *Computer Vision and Pattern Recognition*, 8(3):431 – 440, 2007. doi: 10.1109/CVPR.2005.482. Cited on pages 3, 5, 32, and 43.

[29] Tomer Toledo, David Zohar. Modeling Duration of Lane Changes. *Transportation Research Record: Journal of the Transportation Research Board*, pages 71–78, 1999. doi: 10.3141/1999-08. Cited on page 32.