
Personalized Driving Assistance to Predict Lane Change Manoeuvres

Personalisierter Assistent zur Vorhersage von Spurwechselmanövern

Master-Thesis von Jan Geukes

Tag der Einreichung:

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten: Hien Dang, M.Sc.



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science Department
Knowledge Engineering Group

Personalized Driving Assistance to Predict Lane Change Manoeuvres
Personalisierter Assistent zur Vorhersage von Spurwechselmanövern

Vorgelegte Master-Thesis von Jan Geukes

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten: Hien Dang, M.Sc.

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 08.09.2016

(Jan Geukes)

Abstract

Advanced driving assistance systems can help to increase driving comfort and safety aspects. More and more systems are also using the observations of the driver as an additional input for the models. Especially in the case of prediction of lane changes this personal information are useful. To model this behaviour a Hidden Markov Model is used. In the first part of this work it is shown that the performance of the assistance system for lane change intention detection improves if the model parameters are learned individually. Moreover, the structure of the Hidden Markov Model is also individual for every driver.

The second part explains how the personalized learning can be adapted to an on-line learning approach. For this purpose a model management systems is implemented and two different incremental learning methods are used to update the model parameters during runtime. It could be shown that the same performance like in the off-line scenario could be reached. With the usage of ensemble learning and a weighted majority vote the on-line learning approach shows that even better results are possible.

For the driving assistance system to detect the lane changing intention it is useful to learn personalized models, this can be done in an on-line scenario. The performance is the same as in the off-line learning process and fewer data is needed.

Contents

| | |
|--|-----------|
| 1. Introduction | 5 |
| 1.1. Motivation | 5 |
| 1.2. Problem Setting and Goal of this Work | 6 |
| 2. Theoretical Background | 7 |
| 2.1. Machine Learning | 7 |
| 2.1.1. Expectation Maximization Algorithm | 9 |
| 2.1.2. Evaluation of Models | 10 |
| 2.2. Hidden Markov Models | 11 |
| 2.2.1. Definition and Structure Hidden Markov Model | 11 |
| 2.2.2. Training of Hidden Markov Model parameters | 13 |
| 2.3. Gradient Methods to learn a Hidden Markov Model | 15 |
| 2.4. Elements of Personalization | 15 |
| 2.4.1. Personalization and On-line Learning | 16 |
| 2.4.2. Model Management | 20 |
| 3. Related Work | 22 |
| 3.1. Hidden Markov Models to predict lane changes | 22 |
| 3.2. On-line Learning for Hidden Markov Models and personalization | 23 |
| 4. Data Set and Labels | 25 |
| 4.1. Record Data Set | 25 |
| 4.2. Pre-processing and Data Labels | 27 |
| 5. Methodology | 29 |
| 5.1. Select Model Parameters | 29 |
| 5.1.1. State Selection | 29 |
| 5.1.2. Feature Selection | 30 |
| 5.1.3. Combine Feature and State Selection | 32 |
| 5.2. On-line Learning Algorithms | 33 |
| 5.2.1. Data and Labels | 33 |
| 5.2.2. On-line Evaluation | 34 |
| 5.2.3. On-line Algorithms | 34 |
| 5.3. Model Management | 37 |
| 5.3.1. Model Pool | 38 |
| 5.3.2. Data Management | 41 |
| 5.3.3. Update function | 41 |
| 5.3.4. Model Pool Management | 42 |
| 5.3.5. Prediction and Model Selection | 43 |

| | |
|---|-----------|
| 6. Evaluation and Results | 45 |
| 6.1. Personalization | 45 |
| 6.1.1. Data | 45 |
| 6.1.2. Standard Model Learning | 45 |
| 6.1.3. Feature Selection | 47 |
| 6.1.4. State Selection | 52 |
| 6.1.5. Combined Feature and State Selection | 53 |
| 6.1.6. Summary and comparison | 54 |
| 6.2. Incremental Learning | 56 |
| 6.2.1. Incremental Batch | 57 |
| 6.2.2. Baldi | 57 |
| 6.3. Model Management | 58 |
| 6.3.1. Model Management Parameter | 58 |
| 6.3.2. Model Selection and Prediction | 61 |
| 6.3.3. Environment influence and Data | 65 |
| 7. Conclusion and Future Work | 67 |
| 7.1. Future Work | 68 |
| Appendices | 69 |
| A. Feature Selection | 70 |
| B. Incremental Hidden Markov Models | 71 |
| C. Evaluation | 72 |
| D. Single Voting | 73 |

1 Introduction

1.1 Motivation

Advanced driving assistance systems (ADAS) are useful helpers to increase the security on the streets and help to reduce victims in road traffic. There are many kinds of assistance systems which range from rather simple warning systems during parking to more advanced lane keeping assistance. Additional to the vehicle's dynamics and the environment data also the behaviour of the driver could be used as input for different models. To get this kind of information, for example interior cameras could be used to observe the driver. An interesting case is the prediction of lane changing intention. The driver is preparing the lane change in advanced, a system that knows about this fact could help to warn others and ensure the safety of the manoeuvre. Beside input from driver interactions with the car control instruments, also indirect interactions could be used as input for models, that arises new challenges. First the drivers could be quite different in appearance and behaviour. So potential features should be invariant to the physical stature of the driver. Furthermore, also the behaviour may be different due to a different level of experience or habits. So the style of driving may differ among the drivers and it becomes harder to find a good general model. This leads to the idea that driving assistant should be personalized to the driver.

A good data base is the key factor for a successful model and driving assistant. With personalized models this data is harder to get, because usually nobody knows the specific customer and its behaviour in advance. And a general model on a large data basis would not reach the perfect model performance for every driver, because it is always a trade off between the different driver types.

The idea to overcome these difficulties is, that the model should improve while already using the assistance in the car. In practice, it could be imagined to deliver a product with a general model, this model works already on a good level for most of the drivers. But the system keeps learning and so after a while it is capable to adapt to the characteristics of the specific driver. A further advantage is that the model also can adapt appearing to changes in the future. The personal model should be reached as fast as possible because the amount of data storage and computational power in the car is limited.

The personalization of driving assistance systems is a challenge that affects all kind of systems that use driver specific behaviour as an input signal. Especially situation with lane changes causes a lot of accidents on road traffic. Driving assistance system could work and intervene best if the reaction time before the actual lane change is as long as possible. Maybe the look over the shoulder is done before interact with the car instruments, in this case using the indicator. Therefore, it is useful to consider also the drivers' behaviour in the model. In this context, the development of personal and continuous learning systems could increase the safety on the road for everyone.

In this work the use and the performance of an individual and incremental model learning for lane change prediction is shown. Therefore, an already existing model for detecting the intention to change the lane is used. The basis model works with a Hidden Markov Model to predict the lane changes.

1.2 Problem Setting and Goal of this Work

Based on a given model to predict the lane change intention, this work should examine the following questions.

- How useful is a personalization for this model?
- What are the individual model parameters for changing a lane?
- Could the model be personalized during driving the car?
- How much data is needed to update the model and learn a personalized assistance system?
- How to ensure a certain model quality and not decrease the model performance while learning from new data?

Therefore, environment data, vehicle's dynamics and control input is available as well as the observation of the driver. The driver is watched with a system that delivers features concerning the head movement and the gaze direction. So no pure video data is used as input feature.

Following assumptions and constraints have to be made:

- The system knows exactly the driver of the car.
- The assistant is limited to the highway scenario, so no city or countryside traffic is considered.
- All the data should be processed inside the car, so no cloud scenario where the data is transferred to a central server and processed there, is part of this work.

Although the real hardware configuration and possibilities are not a central point in this work, a solution with limited storage and computing requirements is preferred.

2 Theoretical Background

In this chapter a short introduction, to necessary machine learning parts, is given. In particular Hidden Markov models and its methods are described. Another part explains how to use a Hidden Markov Model to predict lane changes. Aspects on how to extend and personalize these models are described in the last part.

2.1 Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience [32]. This simple principle can be used with many different methods and approaches. But basically machine learning can be divided into three sub categories [2]: Unsupervised Learning, Supervised Learning and Reinforcement Learning. In Unsupervised Learning an interesting question is to find a model that explains the given data set. A problem class in Unsupervised Learning is clustering. Here the goal is to find similarities within the data. To find clusters, algorithms like K-Means or autoencoders are used. Another field of unsupervised methods is dimensionality reduction, here the algorithms try to find a representation of the data that have lower dimensionality (for example Principal Component Analysis) [2].

If additional information in form of labelled data is used, it is called Supervised Learning. An example would be classification or regression tasks. A cost function is used that compares the prediction of the model with the labels of the data, so the goal is to minimize the errors.

Furthermore Reinforcement Learning, here the learning process is iterative and uses the experience of former actions. Therefore a reward function for the different actions is defined. Usually it can be found in robotics or gaming tasks [38]. This work the focus is on Supervised Learning as well as Unsupervised Learning.

Machine Learning Process

In order to solve supervised learning problems in [1] a general process is suggested, which could be found in Figure 2.1.

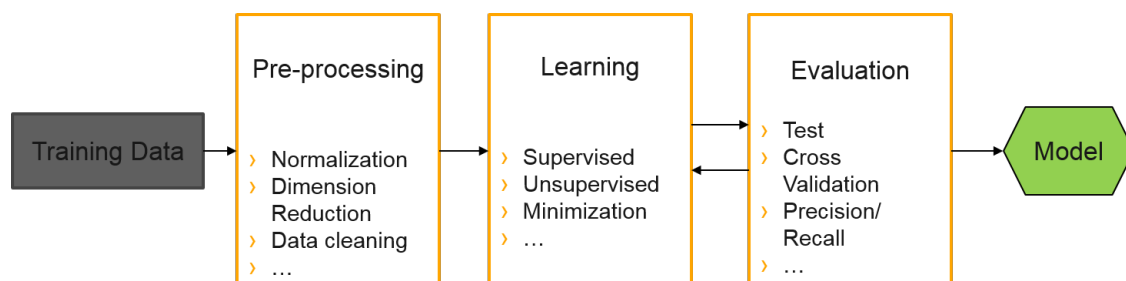


Figure 2.1.: Machine Learning Process [1]

After finding a problem definition, the data set should be analysed. When working with real world data, data pre-processing is necessary in most cases. In this step out-liners are handled

or the data is normalized. After cleaning the data set a suitable algorithm is chosen and a model is trained and then evaluated. If necessary the model parameters have to be adjusted as well, therefore a third unused validation set is necessary to avoid over fitting. As final result a classifier could be implemented which takes new sample points and make a prediction of the class belonging to the new sample point.

Maximum Likelihood

An important function in machine learning is the maximum likelihood function. Considering a density function $p(x | \Theta)$ where Θ is a set of parameters. Furthermore the data set of size N which is drawn from this distribution is given. For example $X = \{x_1, ..x_N\}$. In most case it is assumed that the data vectors are independent and identically distributed (so the data is i.i.d). The density for the samples can be expressed like:

$$p(X | \Theta) = \prod_{i=1}^N p(x_i | \Theta) = \mathcal{L}(\Theta | X)$$

This is a function over the parameter space Θ with a fixed X and called likelihood function \mathcal{L} . In the maximum likelihood problem the best parameter set Θ should be found.

$$\Theta^* = \arg \max_{\Theta} \mathcal{L}(\Theta | X)$$

This leads to an optimization problem which is easier to solve if the log likelihood function is used. Then the product vanished and is replaced by a sum and the numbers become numerical more stable.

Model Complexity and Over-fitting

A common method to evaluate the model complexity is the Bayesian Information Criterion (BIC) this method compares the number of model parameters with the best maximum likelihood result [5].

$$BIC = -\ln(p(x | \theta, M)) + k * \ln(n)$$

The term $p(x | \theta, M)$ denotes the likelihood function for a model M and its parameters θ . Furthermore x defines the observed data and n is the number of samples of this data. With k the number of free parameters to be estimated is described. A lower value in the BIC is preferred.

In general given models with the same performance the model with the lowest complexity should be preferred to avoid over-fitting. This principle is also called Occam's razor [10].

Over-fitting is the problem that the model just represent the training data and do not generalized well to new unseen data. Normally the training data is limited and not a perfect representation of the complete data for this problem [5]. Another approach to deal with limited data and the over-fitting problem is described in the next section.

Cross Validation

A further important concept in Machine Learning is the separation of trainings and test data. Therefore, a part of the data is defined as trainings and another one as test data. Usually the training data set is larger than the test data set. A ratio of 3:1 or 2:1 is common [25]. In the case of not enough data a method like Cross Validation could be used. Here the trainings and evaluation process is done several times but in each iteration the test and trainings sets are changed. So if a model is learnt on the first 3/4 of data and evaluated on the last quarter in the first iteration, in the second iteration the first 1/4 of data is used for evaluation and the rest for training, and so on. At the end the mean value of all evaluations is taken as model performance [5].

2.1.1 Expectation Maximization Algorithm

Expectation Maximization Algorithm (EM) is used to find the maximum likelihood estimate in the case of missing or unknown data. The incomplete but observable data called X and latent variable, which is not observable, is called Z . The set of $\{Z, X\}$ describes the complete data. And the model parameters are denoted by θ . The EM algorithm iterates over two steps which are the estimation step (E-step) and the maximization step (M-step). This leads to the formula for the incomplete data likelihood function [5]

$$p(\mathbf{X} | \theta) = \prod_{\mathbf{z}} p(\mathbf{X}, \mathbf{z} | \theta) \quad (2.1)$$

Because the latent variable Z is unknown, the only knowledge that is available is the posterior distribution

$$p(\mathbf{Z} | \mathbf{X}, \theta)$$

The EM Algorithm now uses the parameter values from the previous iteration θ^{old} to evaluate the posterior distribution $p(\mathbf{Z} | \mathbf{X}, \theta^{old})$. In the first step values of θ^{old} have to be initialized. Then this distribution is used to find the expectation of the complete log likelihood function for some general parameter θ . Hereby the expectation is given by the Q function [5]:

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{z}} p(\mathbf{Z} | \mathbf{X}, \theta^{old}) \ln p(\mathbf{X}, \mathbf{z} | \theta)$$

After performing the E-step, the task of the M-step is to find the optimal parameters for the Q function and so to get the new set of parameters: $\theta^{new} = \arg \max_{\theta} Q(\theta, \theta^{old})$.

The complete EM algorithm then iterates over the E-step and M-step and can be summarized as followed [5]:

- Initialize θ^{old}
- Expected value of the posterior distribution $p(\mathbf{Z} | \mathbf{X}, \theta^{old})$ - E-step
- Optimization of the Q function $\theta^{new} = \arg \max_{\theta} Q(\theta, \theta^{old})$ - M-step

- set $\theta^{old} = \theta^{new}$ and continue with step 2 unless the convergence criteria for the likelihood or parameters is fulfilled.

The algorithm finds a local optimum, but not a global solution to the problem. After learning the model parameters it is necessary to evaluate the model performance, which is described in the next part.

2.1.2 Evaluation of Models

After learning the parameters the question is how to evaluate the model. For classification task a confusion matrix as shown in Fig. 2.2 can be build up.

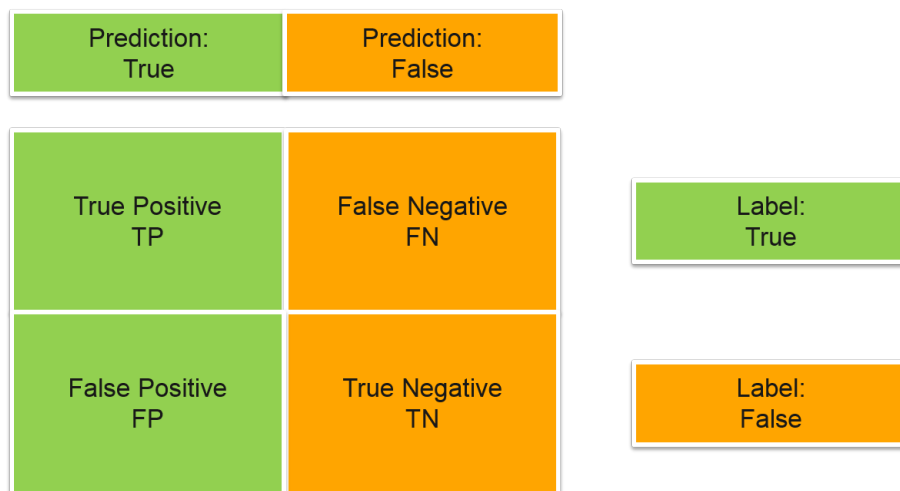


Figure 2.2.: Confusion Matrix for a classification task

This matrix compares the predicted values and the actual values. TP are true positive values while FP are false negative predictions. Accordingly, FP (false positives) and FN (false negatives) are named [30]. To be able to compare different models also the following metrics are defined [11].

- Recall

$$Re = \frac{TP}{TP+FN}$$

- Precision

$$Pr = \frac{TP}{TP+FP}$$

- False Positive Rate

$$FPR = \frac{FP}{FP+TN}$$

- True Positive Rate

$$TPR = \frac{TP}{TP+FN}$$

Depending on the problem just the Recall or Precision metric may not be a good explanation of the model performance. So the harmonic mean of both is introduced and it is called F1 score.

$$F1 = \frac{2 \cdot Pr \cdot Re}{Pr + Re}$$

For an alternative approach the False and True Positive Rate is used. The so called ROC curve (Receiver Operator Characteristic) shows the False Positive Rate on the one axis and the true positive Rate on the other axis. A good model try to optimize the area under this curve. As Davis and Goadrich have shown F1 and Roc are not the same but have some similarities [11].

In this work the F1 score is used because it enables a comparison with previous works and gives a good trade of between precision and recall which shows a good interpretation in this problem setting. Furthermore some adaptations are made to the F1 score. The fact that the manoeuvre lane keeping is performed most of the time causes that lane keeping is not counted as true positive. If doing so a model which predicts just lane keeping would get a high score. Accordingly also the definitions of FP and FN is adapted (Appendix C).

2.2 Hidden Markov Models

Hidden Markov Models can be seen as an extension to Gaussian mixture models and they are used to model hidden or incomplete data. Hidden Markov models are common in speech recognition, handwriting detection or for the analysis of biological sequences such as proteins and DNA [5].

2.2.1 Definition and Structure Hidden Markov Model

A Hidden Markov Model (Model) is a probabilistic model for sequential data. The stochastic process contains two main principles. First there is a Markov Chain which is built from a finite number of unobservable states. They are called hidden states. The second part is an observation probability which is connected to every state. The hidden states are discrete random variables. The observation probabilities could be either discrete or continuous probabilities. This probability is often called emission. To get a Hidden Markov Model working, two assumptions have to be made. First every hidden state depends just on its previous state. So it is independent to all other states. Such a model is called first order Hidden Markov Model. Here O_t is the observation at time t and Q_t is the hidden variable and so the inner state.

$$P(Q_t | Q_{t-1}, O_{t-1}, \dots, Q_1, O_1) = P(Q_t | Q_{t-1})$$

The second requirement is that given a hidden variable the observation just depends on the given variable and not on any other state.

$$P(O_t | Q_T, O_T, Q_{T-1}, O_{T-1}, \dots, Q_1, O_1) = P(O_t | Q_t)$$

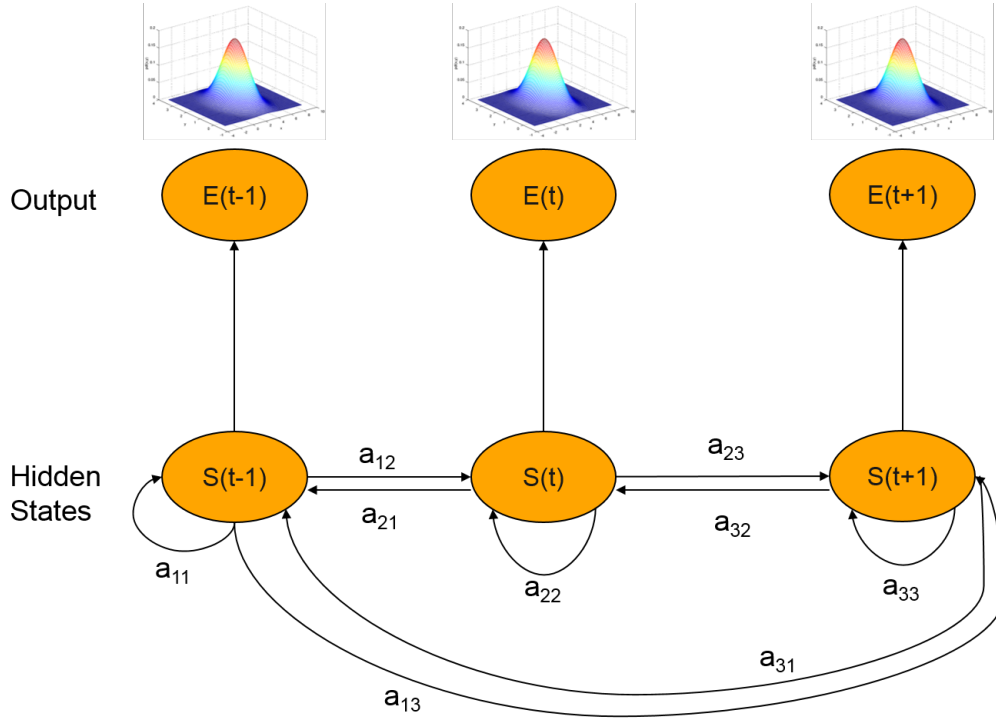


Figure 2.3.: Example of the structure of a Hidden Markov Model with three hidden states and a Gaussian emission probability

A Hidden Markov Model can be described by the following parameters [4].

- Number of hidden states N
- Initialize distribution $\pi = \{\pi_i\}$, every π_i is the probability of starting in state i .
 $\pi_i = P(s(1) = i)$
- Transition matrix $A = \{a_{ij}\}$ gives the probability to change from state i in to j in the next time step: $a_{ij} = P(s(t+1) = j | s(t) = i)$
- Emission matrix $B = \{b_{j(o_t)}\}$, which describes the probability for an input given a certain state.

The function π and A are discrete distributions. But B can also be a continuous distribution. In most cases a Gaussian distribution can be used, like in this work. But this decision depends on the characteristics of the observation. In the case of uni-variant Gaussian, B consist of the mean values and the covariance matrix for each state, $b_j(o_t) \sim \mathcal{N}(\mu_j, \sigma_j)$ for a scalar observation o_t and with μ_j as the mean value and σ_j is the variance. And for the multivariate case the probability density function is defined as following:

$$P(x) = \left(\frac{1}{2\pi}\right)^{k/2} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

It just exists if the covariance matrix sigma (Σ) is semi positive definite. The letter k indicates the dimension of this Gaussian. For each time step these parameters θ can be used to formulate the joint probability of state and emission like the following:

$$p(S, B | \theta) = p(s_0 | \pi) \prod_{i=1}^T p(s_i | s_{i-1}, A) p(b_i | s_i, B) \quad (2.2)$$

where $S = \{s_0, \dots, s_T\}$ and $B = \{b_1, \dots, b_T\}$ [35].

2.2.2 Training of Hidden Markov Model parameters

According to Rabiner there are three main problems which are interesting to solve in the context of HMMs [35].

- (1) How to calculate the probability of an observation sequence, given the model parameters and an observation sequence, $p(O | \lambda)$.
- (2) How could an optimal state sequence be defined given an observation sequence.
- (3) How should the model parameters be chosen to maximize the probability of the observation sequence given the model parameter, $\lambda^* = \arg \max_{\lambda} p(O | \lambda)$.

The evaluation problem (1) and the learning problem (3) will be discussed in more detail in the next part. More information to problem (2) can be found by Rabiner [35].

In order to solve problem (3) first the evaluation problem has to be solved. This can be done with the Forward-Backward Procedure [4].

Forward Procedure

The Forward procedure describes the probability of seeing a certain sequence of observations from time $t = 1$ until time step t and ending in state $s(i)$ at time t , given the model parameters. This value α can be computed recursively using the following steps [4].

1. $\alpha_i(1) = \pi_i b_i(\mathbf{e}_1)$
2. $\alpha_j(t + 1) = \left[\sum_{i=1}^N \alpha_i(t) a_{ij} \right] b_j(\mathbf{e}_{t+1})$

Backward Procedure

Similar to the Forward Procedure the Backward Procedure is calculating a probability for seeing a partial sequence and ending in a certain state. But it is looking backwards at the sequence from the last point T until current time step t . This behaviour is described with the following formula [4].

1. $\beta_i(T) = 1$
2. $\beta_j(t) = \sum_{i=1}^N a_{ij} b_j(\mathbf{e}_{t+1}) \beta_j(t + 1)$

Baum-Welch Algorithm

In order to solve problem (3) the maximum likelihood from formula 2.2 has to be computed. So far there is no way to calculate this property analytical for HMMs [4]. Therefore, it is necessary

to either use a special case of the EM algorithm or use gradient techniques to get an as optimal as possible estimate of the maximum likelihood. The goal is to estimate the model parameters given a set of data and the old model. First the Forward Backward Procedure has to be solved. Then α and β can be used to calculate two more probabilities which are needed to solve problem (3) in the end.

$$\gamma_i(t) = \frac{a_i(t)\beta_i(t)}{\sum_{i=1}^N a_j(t)\beta_j(t)}$$

which is the probability of being in state i at time t given a sequence of observations o . Furthermore ϵ can be defined as $\epsilon_{ij}(t) = p(Q_t = i, Q_{t+1} = j | O, \lambda)$ which describes the probability of being in state i and moving to state j in the next time step. This can also be written as:

$$\epsilon_{ij}(t) = \frac{\gamma_i(t)a_{ij}b_j(o_{t+1})\beta_j(t+1)}{\beta_i(t)}$$

This first part of the Baum-Welch Algorithm is the equivalent to the E-step of the EM Algorithm. In the M-step the new properties to re-estimate the parameters of the Hidden Markov Model can be used. In general following formula could be used to update the parameters [35].

- Start probability: $\tilde{\pi}_i = \gamma_i(1)$
- Transition matrix: $\tilde{a}_{ij} = \frac{\sum_{t=1}^{T-1} \epsilon_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$

The update rule for the emission probability is dependant of the form of this quantity. In the discrete case, with finite and discrete alphabet $V = \{v_k\}$ and an observation o_t the following is used:

- Emission probability: $\tilde{b}_i(k) = \frac{\sum_{t=1}^{T-1} \delta_{o_t, v_k} \gamma_i(t)}{\sum_{t=1}^T \gamma_i(t)}$

And in the case of continuous univariate Gaussian emission probabilities the update rules have to be changed for the parameters:

$$\tilde{\mu}_i = \frac{\sum_{t=1}^T \gamma_i(t) o_t}{\sum_{t=1}^T \gamma_i(t)}$$

$$\tilde{\sigma}_i^2 = \frac{\sum_{t=1}^T \gamma_i(t) (o_t - \mu_i)^2}{\sum_{t=1}^T \gamma_i(t)}$$

For further update methods see [23].

2.3 Gradient Methods to learn a Hidden Markov Model

Next to the Expectation Maximization algorithm the problem of learning the model parameters also can be handled with optimization techniques, like gradient descent or Quasi-Newton methods [23]. These approaches try to optimize directly the maximum likelihood function from 2.1. At the beginning the model θ^0 uses a random guess as initial values. In every iteration k first order methods like gradient descent update the model parameters.

$$\theta^{k+1} = \theta^k + \eta_k \nabla_{\theta} \mathcal{L}_T(\theta^{(k)})$$

where η_k is the learning rate and $\mathcal{L}_T(\theta^{(k)})$ stands for the log likelihood. The learning rate decreases monotonically over the iterations to ensure that the next likelihood function is non-decreasing. The learning rate tries to maximize the objective function in search direction.

$$\eta_k = \arg \max_{\eta > 0} \mathcal{L}_T[\theta^k + \eta_k \nabla_{\theta}(\theta^{(k)})]$$

To overcome problems with the convergence rate in a large optimization space also second order techniques could be used. Here the Hessian matrix H is used and guarantees faster convergence, like the Newton-Raphson method.

$$\theta^{k+1} = \theta^k - H^{-1}(\theta^{(k)}) \nabla_{\theta} \mathcal{L}_T(\theta^{(k)})$$

This works well if the Hessian is invertible and negative semi-definite. Because of the random initialization these properties could not be ensured, so Quasi Newton methods are used. In this case an approximation of the Hessian is used, also this process can be used with a learning rate [23].

2.4 Elements of Personalization

Creating models in the automotive world which including the drivers' behaviour comes along with the problem that every human and its behaviour could be very different. Most models are getting better with more training data. So to get good models which are modelling the drivers' behaviour, a lot of data for every possible driver would be needed. Because in reality this is not likely to have, one approach is to learn a general model that fits best in most of the situations. Another approach is to start with a model that might not be perfect but try to improve it while using it. In this case the model can adapt to the specialities of the driver and get a personalized model for every driver after some time. This idea also raises some new questions. First how to update the model, which parts of the model are individual and which parts are more general. Updating the model during runtime also has the risk of over-fitting. If the model is optimized perfectly to fit old data and the new data block is different the model performance may be even worse than before. So strategies are needed to ensure a certain minimum quality of the model. Furthermore a decision has to be made how fast and often a model can be updated, considering that the data storage and the computational capacity in a car is limited. In the next section first the possibilities for updating a model continuously are described followed by a part about model management which should ensure a stable prediction quality.

2.4.1 Personalization and On-line Learning

To personalize a model there are at least two different ways of looking at this problem. First it is possible to adapt the model parameters to the individual driver. Therefore an algorithm that can incorporate new data of the driver to the model is needed to learn more accurate model parameters. As a second important point is the model itself and the question if some hyper parameters of the model are individual for every test person. For example are the same number of states needed to model the behaviour of a certain driver or the importance of features is different when dealing with different persons?

One way to learn individual model parameters over time is the so called on-line learning. According to Shalev-Shwartz [36]:

On-line learning is the process of answering a sequence of questions given knowledge of the correct answers to previous questions and possibly additional available information.

This means there are several rounds and the model should be able to provide an answer at every time step. It is supervised learning because the information about the right predictions are used to train and adapt the model. Furthermore there are several forms of on-line learning which are defined in the next section.

Different types of on-line learning

There are different forms of learning models in a personalized way. Because of not having enough data for each (driver) model, it is necessary to include new appearing data into the model. Therefore three different strategies could be defined. First *Batch-Learning*, after every new data block a new model is learned completely from the beginning. The new data is added to previous data, so more and more data has to be saved. In opposite *On-line Learning* is using every new data point to update an existing model, so previous information are still used. Nevertheless, just one data set is needed for each point in time. A third way to look at this problem is *Incremental Learning*. It is similar to on-line learning but it rather uses data blocks which can obtain several data points. Incremental algorithms can iterate over the data block until it reaches convergence. So each data point within a data block is used several times. This kind of procedure has got the advantage to be more robust [22]. A graphical overview is given in Figure 2.6.

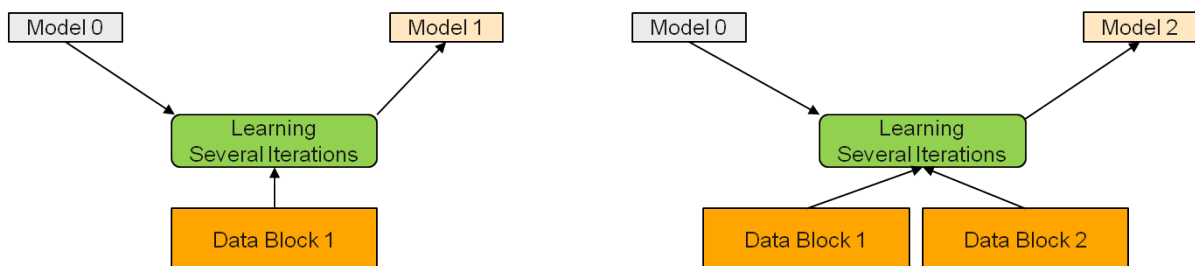


Figure 2.4.: Standard Batch Learning: The data pool is increasing over time. At every time step the model is learnt from the beginning

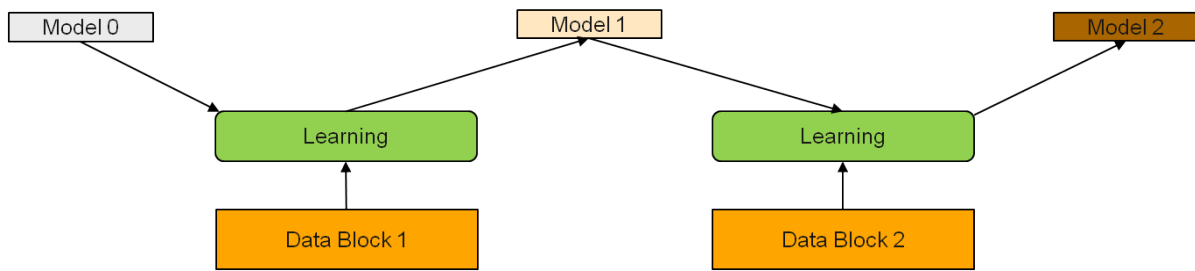


Figure 2.5.: On-line Learning: The model parameters are updated with every new data point, just the model parameters are saved.

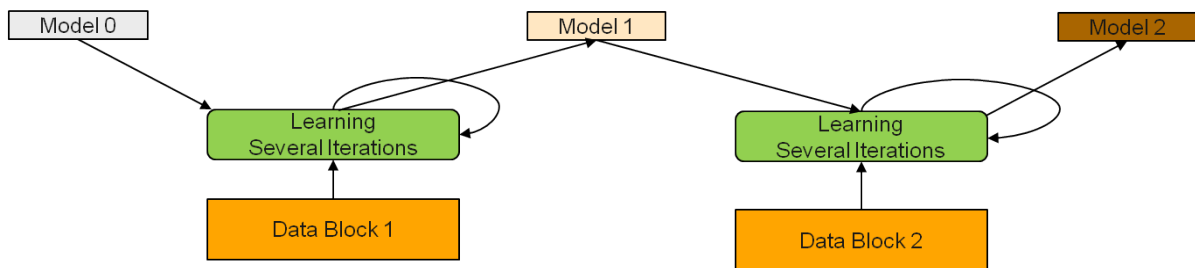


Figure 2.6.: Incremental Learning: The learning process can iterate over a block of data several times like in batch learning. The model parameters are saved and used as a start point in the next iteration.

A further important distinction for algorithms in this field is whether they are working on data blocks (block-wise) or updating the model sample per sample (symbol-wise) [23]. For incremental learning, block-wise algorithms are needed and a data block with length one is a special case of block-wise learning and the same as symbol-wise learning.

In a survey paper by Khreich et al. the methods are also distinguished by the type of the cost function (Minimum Model Divergence, Maximum Likelihood Estimation or Minimum Prediction Error), as well as the optimization technique (Optimization, Expectation Maximization, Recursive Estimation). The complete matrix can be seen in appendix B.

Important to notice is that many incremental or on-line learning approaches for Hidden Markov Models are focused on a special case of HMMs. Restriction to discrete emission probabilities or single continuous Gaussian are common. In the context of speech recognition it is useful just to update the transition matrix [13], [37].

According to the performance in paper "A Comparison of Techniques for On-line Incremental Learning of HMM Parameters in Anomaly Detection" by Khreich et al. two algorithm are chosen for an explanation in detail [22]. As a symbol-wise candidate the algorithm of Cappé [7] is examined and for the block-wise gradient techniques the approach of Baldi and Chauvin [3] is interesting.

On-line Learning: Baldi

The incremental learning algorithm of Baldi and Chauvin tries to optimize the maximum likelihood function using optimization techniques [3]. In this particular case it is inspired by gradient descent. The method needs blocks of data to perform a block-wise update of the model. The

original Baldi algorithm is limited to Hidden Markov Models with discrete emission functions. First the update of the transition matrix is shown and in the second part the update rule for the multi-variant Gaussian from another paper is integrated.

On every new data block the Forward-Backward method is applied. This method works like a filter about the belief of the current state. Then a gradient update of the transition matrix can be performed. To ensure that the resulting matrix holds just positive values between zero and one, a soft max parametrization is used [3].

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_{k=1}^N e^{u_{ik}}}$$

This new introduced parameter u then is updated in every time step of the algorithm. The combination of the described steps leads to following update rule for the transition matrix. Thereby it is defined:

$$\epsilon_{t|T}(i, j) = \frac{\gamma_{t|t(i)} a_{ij}}{\sum_{i=1}^N \gamma_{t|t(i)} a_{ij}} \gamma_{t+1|T}(j) \quad \text{and} \quad \gamma_{t|T}(i) = \sum_{j=1}^N \epsilon_{t|T}(i, j)$$

To gain this information the Forward Backward Algorithm is used. Then the update rule for the new parameter u is the following.

$$u_{ij}^{(k+1)} = u_{ij}^{(k)} + \eta \sum_{t=1}^T (\epsilon_{t|T}(i, j) - a_{ij} \gamma_{t|T}(i))$$

η defines a learning rate to make the learning more smooth.

Baldi uses the same idea for the discrete emission matrices, but in our case an update method for a multi-variant Gaussian is needed [3]. Therefore, an idea of Krishnamurthy et al. is used to update the emission parameters with a gradient method [26]. Like in the previous step, first the information of the Forward-Backward procedure are used and then with the difference of the current and the past model state, the parameters are adjusted.

First the mean values are updated.

$$\mu_i^{t+1} = \mu_i^t + \frac{\gamma_{t+1}(i)(o_{t+1} - \mu_i^t)}{\sum_{\tau=1}^{t+1} \gamma_{\tau|t+1}(i)}$$

In a second step also the covariance matrix is updated with a gradient descent technique.

$$(\sigma^2)^{t+1} = \Sigma^t + \frac{\sum_{i=1}^N \gamma_{t+1}(i)(o_{t+1} - \mu_i^t)^2 - \Sigma^t}{t+1}$$

Another way of solving the problem of learning the model parameters of the Hidden Markov Model On-line is to try to implement an incremental version of the EM algorithm. Cappé suggested an incremental version of the EM algorithm which is working on single data points and tries to optimize the maximum likelihood function [7]. The key idea of the algorithm is to use first a filter update to determine the current state. In the next step the sufficient statistics of the model parameters are estimated and updated. The stochastic approximation is referred as E-Step of the EM methods and the update of the new parameters is called M-step again. Some pseudo code can be found in Fig 2.7.

Initialization Select $\hat{\theta}_0$ and compute, for all $1 \leq i, j, k, \leq m$ and $0 \leq d \leq 2$,

$$\begin{aligned}\hat{\phi}_0(k) &= \frac{\nu(k)g_{\hat{\theta}_0}(k, Y_0)}{\sum_{k'=1}^m g_{\hat{\theta}_0}(k', Y_0)}, \\ \hat{\rho}_0^q(i, j, k) &= 0, \\ \hat{\rho}_{0,d}^g(i, k) &= \delta(i - k)Y_0^d.\end{aligned}$$

Recursion For $n \geq 0$, and $1 \leq i, j, k, \leq m$, $0 \leq d \leq 2$,

Approx. Filter Update

$$\hat{\phi}_{n+1}(k) = \frac{\sum_{k'=1}^m \hat{\phi}_n(k')\hat{q}_n(k', k)g_{\hat{\theta}_n}(k, Y_{n+1})}{\sum_{k', k''=1}^m \hat{\phi}_n(k')\hat{q}_n(k', k'')g_{\hat{\theta}_n}(k'', Y_{n+1})},$$

$$\text{where } g_{\hat{\theta}_n}(k, y) = \exp[-(y - \hat{\mu}_n(k))^2/2\hat{\nu}_n].$$

Stochastic Approximation E-step

$$\begin{aligned}\hat{\rho}_{n+1}^q(i, j, k) &= \gamma_{n+1}\delta(j - k)\hat{r}_{n+1}(i|j) + (1 - \gamma_{n+1})\sum_{k'=1}^m \hat{\rho}_n^q(i, j, k')\hat{r}_{n+1}(k'|k), \\ \hat{\rho}_{n+1,d}^g(i, k) &= \gamma_{n+1}\delta(i - k)Y_{n+1}^d + (1 - \gamma_{n+1})\sum_{k'=1}^m \hat{\rho}_{n,d}^g(i, k')\hat{r}_{n+1}(k'|k),\end{aligned}$$

$$\text{where } \hat{r}_{n+1}(i|j) = \hat{\phi}_n(i)\hat{q}_n(i, j) / \sum_{i'=1}^m \hat{\phi}_n(i')\hat{q}_n(i', j).$$

M-step If $n \geq n_{\min}$,

$$\begin{aligned}\hat{S}_{n+1}^q(i, j) &= \sum_{k'=1}^m \hat{\rho}_{n+1}^q(i, j, k')\hat{\phi}_{n+1}(k'), \\ \hat{q}_{n+1}(i, j) &= \frac{\hat{S}_{n+1}^q(i, j)}{\sum_{j'=1}^m \hat{S}_{n+1}^q(i, j')}, \\ \hat{S}_{n+1,d}^g(i) &= \sum_{k'=1}^m \hat{\rho}_{n+1,d}^g(i, k')\hat{\phi}_{n+1}(k'), \\ \hat{\mu}_{n+1}(i) &= \frac{S_{n+1,1}^g(i)}{S_{n+1,0}^g(i)}, \\ \hat{\nu}_{n+1} &= \frac{\sum_{i'=1}^m (S_{n+1,2}^g(i') - \hat{\mu}_{n+1}^2(i')S_{n+1,0}^g(i'))}{\sum_{i'=1}^m S_{n+1,0}^g(i')}.\end{aligned}$$

Figure 2.7.: On-line learning for HMM according to Cappé [7]

2.4.2 Model Management

All the previous described methods do not guarantee a global optimum either for the current data block or for all the next new data blocks. So it is possible that the model performance decreases after an update step. In this case it is helpful to store the older model, as some kind of fall back strategy, see Fig 2.8. The idea of having a model pool with many models came up. This method is also called Ensemble Learning. [12]. With a model pool, there are also new tasks to deal with in order to just save the promising models in a long term view and do not use too much data storage. Managing the ensemble of different models is known as Model Management. A general overview about such kind of Model Management can be seen in Fig 2.8.

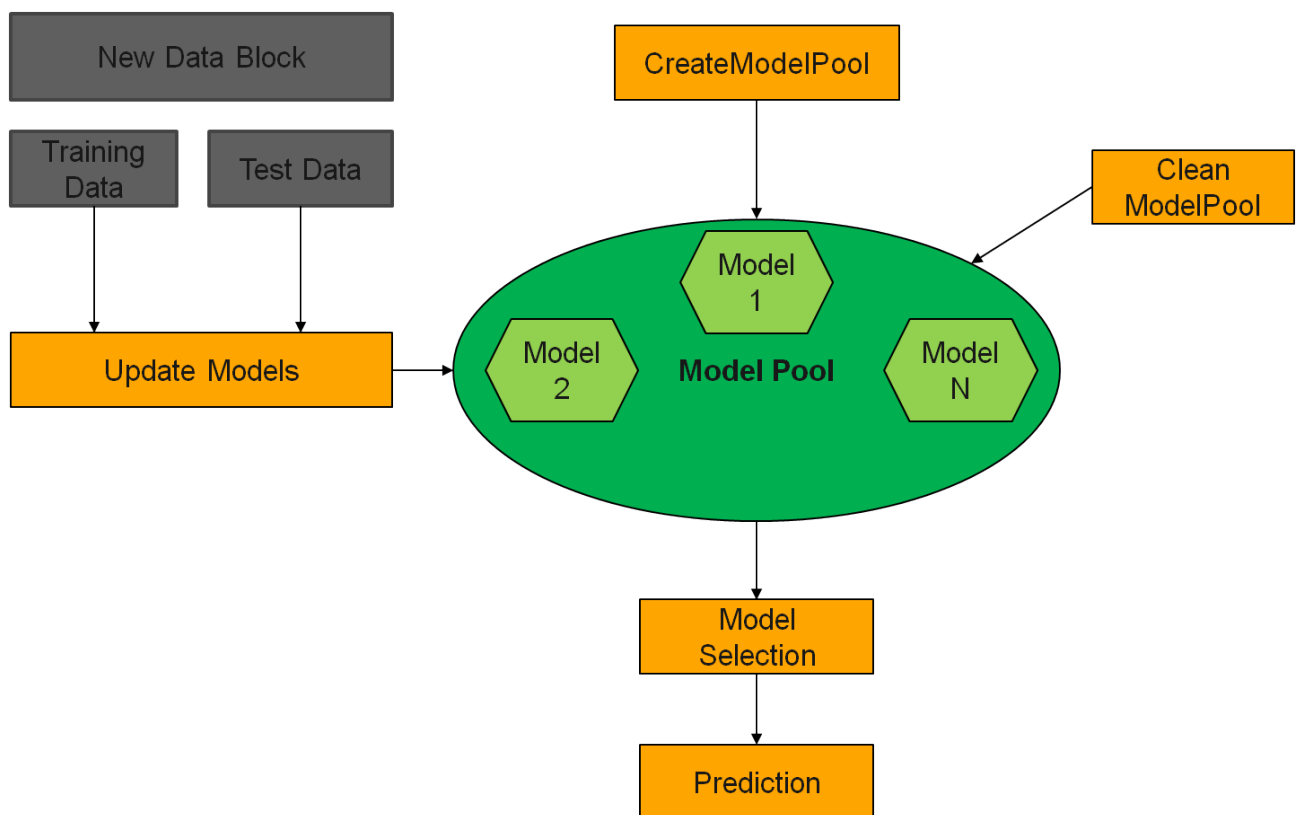


Figure 2.8.: Components of a Model Management after [24]

Key element of this structure is the model pool which holds all the current models. The model pool has to be initialized with a certain number of models. The part "createModelPool" handles this issue. In many cases it is useful that the models inside the pool are not too similar. Furthermore a strategy has to be implemented how to update the models in the pool. This method uses the new data, it needs two blocks of data, one for training and one for evaluation. Models which perform worse than others over a longer time period should probably delete, this is the task of "cleanModelPool". An on-line system should produce answers to every time step and the selection method decides which models of this pool should do the prediction. The criteria for selection and the number of selected models can vary [24].

Ensemble Learning

This part should explain, what are "Ensemble Classifiers" and why they can produce better results than a single classifier. Ensemble methods use the prediction outcome of several single models. Then the results are combined to a new prediction. This could either be a weighted or unweighed voting process. There are two conditions under which an ensemble classifier is more accurate than the single members of this model pool. First the single models in the pool have to be accurate. In theory, they have to be better than guessing, but better basis classifiers may also increase the overall performance of the ensemble. The second important point is, that the members of the ensemble have to be diverse. This means that the single models make different errors on new incoming data [18].

There are three reasons why ensemble can produce better performance scores than single classifiers [12].

- Statistical reason: With limited amount of training data the classifier may not find the true hypothesis function for the problem. But if several hypotheses with similar performance are used, the average over all these hypotheses reduces the risk of choosing the wrong hypothesis.
- Computational reason: Many classifiers work by doing some kind of local search and just find a local optimum. Which local optimum is found depends also on the starting value which may be initialized randomly. So averaging over classifiers that started from many different starting points can provide a better approximation of the true, unknown function for the classifier.
- Representational reason: In many cases the true function for the classification problem cannot be represented by the hypothesis in the search space. So a weighted sum of many may expand the space of representational functions. This is just true if a limited amount of data is assumed and so just a limited amount of hypothesis can be represented.

3 Related Work

This work is dealing with the question of how to personalize the prediction of lane changes. Therefore, the one of the main parts of the related work chapter is covering several approaches of modelling the lane change manoeuvre. Especially the one that has been used in a previous work by Faller which is the basis of this work [16]. In the second part a short overview about related work in the fields of personalization and on-line learning for HMMs is given.

3.1 Hidden Markov Models to predict lane changes

The task to predict lane changes is a highly time dependant classification problem [34]. Hence, there are different kinds of models that are used to provide solutions. On the one hand neural networks and on the other hand Hidden Markov Models. Also, approaches that use Support Vector Machines (SVM) or variations of it have been used to solve this multi-class classification problem. Just using features from lane detection and further driving information [28] presents a solution which uses a SVM. RVM (relevance vector machines) are a specialization of support vector machines and are also implemented for predicting lane changes [34]. This work rather concentrates on the on road design of the method.

Ding et al. are using neural networks to predict the trajectory of a possible lane changes and incorporate the uncertainty of the human factor [14]. Tomar et al. also try to predict the trajectory of the lane change manoeuvre and not just classifying the manoeuvre. In their approach also a neural network is used. This task is much more complex than pure classification problem [39]. A recurrent neural network is used in [21], in contrast to the approach in this work also raw video data is used.

Next to standard Hidden Markov models like in [29], also adaptation of HMMs are used. The Autoregressive Input-Output HMM (AIO-HMM) uses two additional layers to represent internal and external features [20]. Based on an idea of Lefevre et al., Faller has implemented a solution for this multi-class classification problem which is described in more detail in the next part. Additional to [29] it also uses head and eye movement features.

In a Hidden Markov Model the states are not observable, so there is no direct information which state produced the observable output. Therefore a single HMM is trained for every possible outcome class. In this case: it is change lane to the right, to the left or drive straight. The training is done just with the data labelled according to the manoeuvre. In a second step these three sub-models are combined to one bigger HMM with the advantage of knowing which states should predict which output label. The ideal is visualized in Fig 3.1.

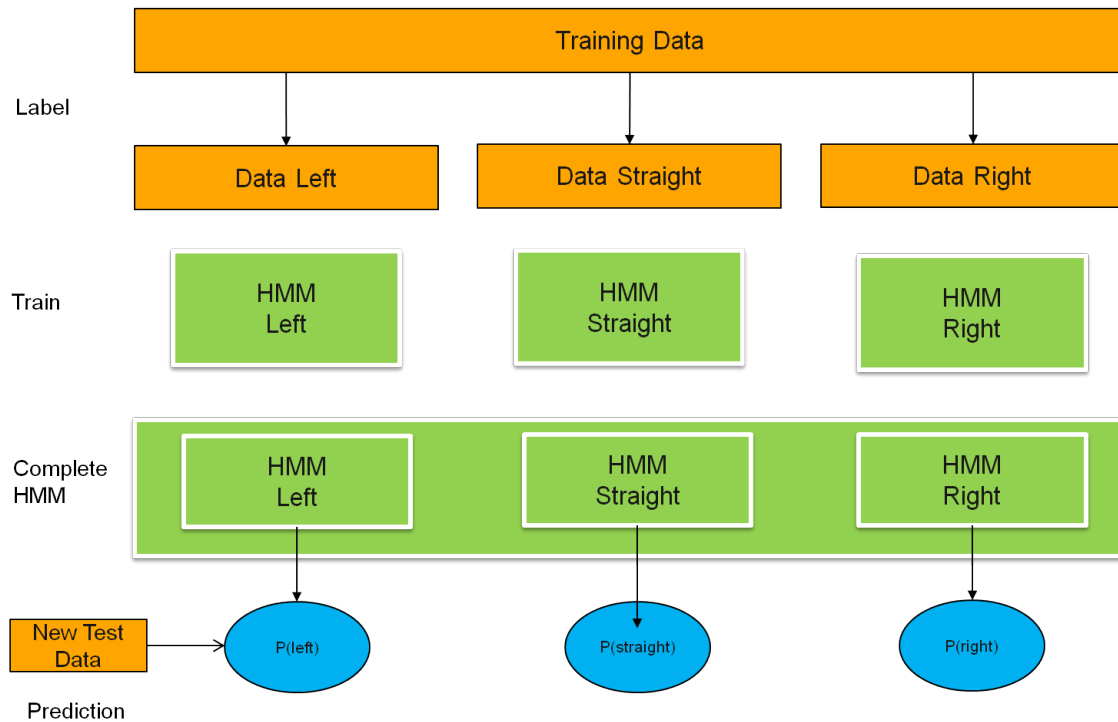


Figure 3.1.: Lane Change Classification with a multi-class Hidden Markov Model

In Faller's work every sample point is used to make a new prediction [16]. The data set is a ride in a driving simulator. 36 different drivers have been examined and every driver drives about 30 minutes. In order to evaluate the models the F1 score is used in a special variant, in which the true prediction of "stay at the lane" is not taken into account of the model performance score (see C).

Before starting the model learning the data is labelled therefore every lane change is detected and the start of the intention for a lane change is defined. The training process of the sub models is done with the Baum-Welch Algorithms, which is a special case of the EM-Algorithm. The complete model then is also trained with help of the Baum-Welch Algorithm but the emission probabilities are fixed and mainly the transition matrix is learned.

3.2 On-line Learning for Hidden Markov Models and personalization

Hidden Markov Models are also widely used in other applications. Therefore there are large varieties of different on-line or incremental methods. This related work part focus on on-line learning methods for Hidden Markov Models. A good overview over the different methods can be found in a survey paper from Khreich et al. [23]. They made an overview of about 22 different approaches (see taxonomy in the appendix B). The field of use differs with the methods. Several papers deal with on-line updates of Hidden Markov Models in the context of speech recognition [37], [13]. In speech recognition the semantic meaning of the Hidden Markov Model is special. So to adapt to different speakers just the emission probabilities have to be learnt. The transition probabilities are derived from the grammar of the language that is used.

Many use cases also deal with symbolic output emissions. This discrete cases are easy to update [33], [17]. An area of application for such kind of Hidden Markov Models is handwriting

recognition [41]. The survey describes the theoretical properties of the learning algorithm. A performance benchmark on real world problems could not be found. For single approaches the work of Cavalin et al. shows a comparison of different types of incremental learning approaches [8] and in different scenario a comparison can be found also at [22].

Another application field for on-line HMMs is the anomaly detection in computer networks. This use case is more common to the problem in this work also the relevant events can be very rare in the network setting [24]. In this work also ensemble methods are used and a model management is introduced.

In the context of ensemble methods also other adaptive learning approaches can be found. The on-line Passive-Aggressive Algorithms try to learn just when the model do not predict right, in a supervised scenario [9].

So far most approaches try to update the parameters of the HMM. In [40] the complete HMM is changing and growing, so also states and features configurations are updated.

4 Data Set and Labels

There is a data set that already exist and is used in the work of Faller [16]. The data set contains 36 test persons and was recorded in a driving simulator. Every driver had to complete a track, which takes about 30 minutes. The high number of drivers supports a greater variety in the behaviour of the drivers and is a good source to learn a generalized model. The main disadvantage is that there is just a relative short time of driving per driver that is the reason why it is not a perfect data set to learn individual models. In this 30 minutes drive the average number of lane changes is about 60. For a general model in Figure 4.1 it can be seen, that after about 150 lane changes the model has been converged.

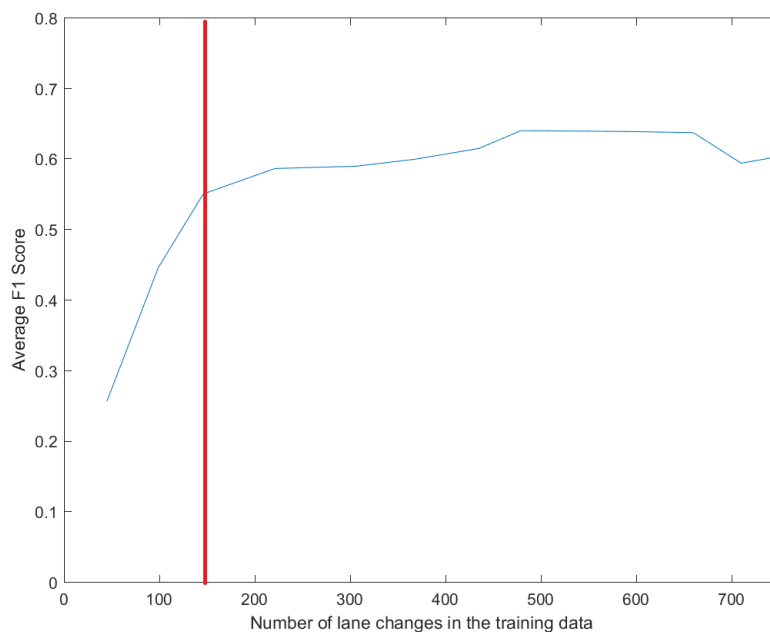


Figure 4.1.: Development F1 Score compared to the number of lane changes in the training data

So even if maybe slightly less lane changes are needed to learn an individual model there are not enough lane changes per driver available in the current data set. Therefore in the next section it is explained how a new test run has been designed to overcome this problem. Again the setting is limited to the highway driving situation. Furthermore this chapter contains a section about the data label and data pre-processing steps.

4.1 Record Data Set

The goal is to create a new data set which is more suitable to answer questions concerning personalization. Therefore enough lane changes per driver are needed. A rough estimation on how many lane changes are needed could be found in Fig. 4.1. This calculation is true under the assumption that the personalized model and the general model have a similar convergence rate. This means more than 150 lane changes per driver are desirable. It is just possible to gain

the new data in the driving simulator and not in a real car because an appropriate equipped car is not available. It is known that driving in a simulator for a longer time can cause simulator sickness [6]. Therefore each session in the simulator should be as short as possible. In a former study 2 lane changes per minute have been reached. So the experiment has been designed in two sessions with about 45 minutes each to reach 180 lane changes per driver.

In the simulation environment it is possible to create a track that forces the driver to do lane changes. Therefore, a highway with 2 lanes in each direction is used. Just the direction of the test car is interesting for the next sections. On the right side of the road the other cars are driving slower than the current speed limit. So the test driver is starting on the right lane and has to overtake the slower cars in front of him. To avoid that the driver is just driving on the left lane following measurements are implemented:

- The distance between the slow cars on the right lane is big enough to move back on the right lane again, especially if the driver is obligated to drive on the right side of the road (according to German traffic rules).
- On the left lane also other cars appear randomly and drive with a much higher speed, so that the test driver is "forced" to give the left lane free again.
- The driver is encouraged to follow the general traffic rules.
- An acoustic signal remains the driver if it is above the current speed limit.

The foreign cars on the left lane also make sure that the driver has to pay attention before changing the lane and check its environment. More and faster cars on the left lane makes the overtake process harder, so it is also a way to regulate the difficulty of the task. The complete situation is illustrated in Fig 4.2.

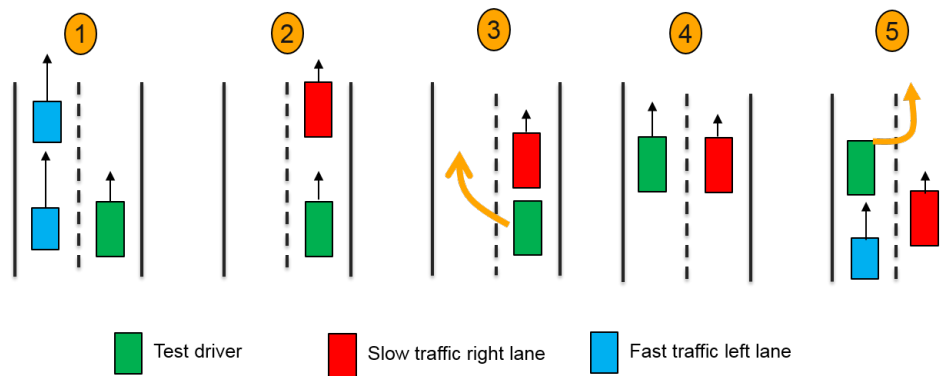


Figure 4.2.: Figure shows the steps of the lane change manoeuvre and how to enforce more overtake manoeuvres

The traffic flow on the right and left lane is generated automated and the cars appear in a certain distance with Gaussian noise added.

Test course

The final test course contains four different course sections as can be seen in Table 4.1. Three of the sections have speed limit 130 km/h and another one has a limit of 80 km/h.

Table 4.1.: Different test run sections

| Id | Level | Speed Limit | Description |
|----|----------|-------------|---|
| 1 | easy | 130 km/h | no traffic on the left lane |
| 2 | moderate | 130 km/h | moderate traffic on the left lane |
| 3 | hard | 130 km/h | a lot of traffic on the left lane |
| 4 | moderate | 80 km/h | slower traffic on the right and moderate traffic on the left lane |
| 5 | relax | 130 km/h | no traffic at all |

Furthermore the traffic on the left lane is regulated from easy to heavy. The complete course starts on a highway rest area and also ends there. Every section is used twice per session and after seeing every part once there is a short relaxing section without any traffic at all. The order of the main sections differ like in Figure 4.3.

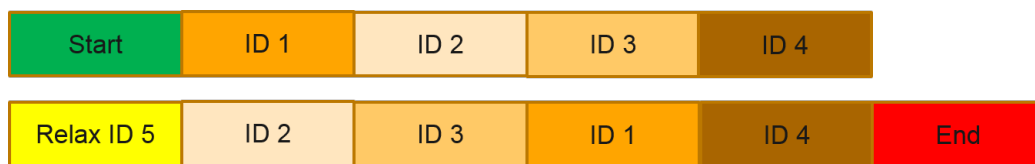


Figure 4.3.: Different sections of the test course. Detailed description for each section could be found in the table above.

Recorded Data

The simulator environment is recording the data with 100 Hz. There are about 300 data streams which can be divided in following areas:

- Information about the car state itself, like speed, acceleration, wheel torque.
- Information about the direct environment like line detection, position of the car in relation to the road and about other cars next to it and its distance.
- Information about the course itself, for example which course section and the time.
- Information from the head movement and eye tracker. The driver is monitored with interior cameras, but just the aggregate information are available not the plane video streams. This system provides a quality measurement for certain features.

For reasons of validation also the driver itself is recorded with a video camera, but this information is not used to create the model.

4.2 Pre-processing and Data Labels

The learning process is done in supervised way and so the training data needs to be labelled. This should be done in a automated way. The problem of predicting the intention to change the lane is, that except the driver itself nobody really can be absolutely sure about current intentions. But there are two steps how a prediction label still can be set. First a lane change is done when

the car is crossing the middle line. Modern driving assistance systems can detect the lines on the road quite well [31]. After knowing the end point of the lane change manoeuvre the next step is to find the start of the intention. Therefore the algorithm is going back in time and is looking for the strong head movements in a certain interval. This first strong head movements indicates in most cases the active start of looking back to check whether other lane is free. The minimum time for a lane change is two seconds and the maximum time is six seconds. In average the lane change got the labels 3.1 seconds in advance of the actual change. Alternative the interval of the lane change label also can be set to a fixed number. But the dynamic method produces more realistic results and is therefore used during this work.

Even the data produced by the simulator needs some cleaning and pre-processing in advance. For head and eye movement data a quality signal is available which could be used to deal with bad measurements. Because the corrupt data usually just appears for a few time points it is possible to replace values where the quality measurement is below a threshold with the previous values.

Moreover, for example the feature that shows the distance from the driver's car to the next car driving in front needs some adjustment. Cars in front of the own car can just be detected in a certain range, if there are no cars in this range the system returns 0. This is misleading because the value of having no car in front of the own car is quite close to the situation where a car is right in front of the driver's car. In order to model this data as a Gaussian the values of the no car in front situation are shifted out of the normal range.

The time that it takes to drive on and off the highway at the beginning and end of the test run is cut out of the data stream.

New Data set

The suggested test course from graphic 4.3 have been implemented and test runs with six different persons have been done. Every driver was driving in two sessions with a length of about 40 to 45 minutes. During this time the drivers produce about 2 lane changes per minute. This experiment focused on bigger data set per driver and not so much about a high variety of different drivers. In the questionnaire the experience of driving per year have been asked. Here the range of the answers have been from less than 500 km up to more than 5000 km. Particularly when compare the personalized learning with a general model the degree of similarities of the drivers would be interesting, but due to time limits this could not be tested in detail. Nevertheless, the new data set which is used in this work is very similar to the one used before (also in [16]) so that results still could be compared to a reasonable extend.

5 Methodology

This chapter describes methods and processes, which are used in this work. The chapter is split into two parts. The first part is about the question of how useful is it to learn a personalized model? And if it is useful to learn the models individual, in which parameters the personalized driver models differ? To answer these questions the complete data set per driver is used and the models are learnt off-line. The second part deals with an on-line scenario. Here short data blocks arrive at every time step. These data blocks should be used to learn a personalized model during run-time. To ensure the quality during this process a model management system is implemented.

5.1 Select Model Parameters

There are two kinds of model parameter for HMM. First parameters which describe the structure of the Hidden Markov Model. The structure of the HMM is defined by the number of hidden states and by the set of features which are used as input signals. These parameters are fixed during the training process. The second category of model parameters contains the parameters that are learnt during the training process. In the case of HMMs these are the transition matrix and the emission probabilities.

The trained parameters describe the shape and the relation between the given structural parameters. These parameters can also be individual, for example the strength of the head movement may be different before a lane changing manoeuvre. So the EM algorithm is used to train the model for every driver separately. Like described in [16] first the data is split according to the labels (change lane to the right, change lane to the left or stay at the lane). Then a sub model is trained for every manoeuvre type and finally the sub models are combined to one model. So it is possible to classify the three different classes with the HMM. The methods to learn the parameters in this setting have already explained in the work of Faller [16].

Also structural parameters may differ from driver to driver, so a different number of states could be needed to represent behaviour while staying straight at the lane. Also not every feature may be important for every driver. As an extreme example for a driver who is not looking into the mirrors the head movement feature is not relevant. In the following section methods are explained how to find these personal structural parameters for each driver. First the search for the right state configuration is explained and then a section about the feature selection process is following.

5.1.1 State Selection

The task is to find a suitable number of hidden states for each driver. Therefore the number of states for three sub models has to be chosen. The sub models are defined by '*lane change left*', '*lane change right*' and '*stay straight*'. For the state configuration '*stay straight*' it is expected to need more states than for the two other configuration parameters because most of the time the driver is driving straight and here a higher variance is expected.

The algorithm iterates over all possible combinations of state configuration. Then for every

Table 5.1.: Search space state selection

| | Lane Change Left | Stay Straight | Lane Change Right |
|--------------|------------------|---------------|-------------------|
| Search Space | 1-5 | 1-15 | 1-5 |

configuration a model is trained given a certain feature set. As a next step all models are evaluated on an extra validation data set. And the F1 score is used to determine the best state configuration for each driver. The feature set is chosen from a previous work which should have good generalizing properties [16]. Because this process is relative time consuming the search space is limited for each variable, this information is given in Table 5.1.

For computation and robustness reasons it is better to reduce the model complexity, so rather models with fewer states should be used (see Occam’s razor 2.1). Common metrics to evaluate the model complexity take the number of samples into account. The high amount of samples for the problem in this work leads to metrics that always prefer models with higher number of states. This metrics cannot used to reduce the model complexity. So another simple metric for the model complexity is used to depict the trade of between model complexity and model performance. And also to make sure that with a certain number of states a good performance is reached. Adding many more states may just give a very small performance gain.

Therefore a mean F1 score of all state configurations during the search is calculated. Then the ratio between the deviation of the value from the mean value and the summed number of states of the configuration is used. For every state configuration a new score is calculated like this:

$$Score(configuration) = \frac{F1_{configuration} - Mean(allConfigurations)}{StatesLeft + StatesStraight + StatesRight} \quad (5.1)$$

For every driver the data set is split up. Half of the data is used for training the model. One quarter then is used to validate the model and decide which one is the best configuration. This decision then is tested on the last quarter of the data set.

5.1.2 Feature Selection

There are about 300 features recorded by the driving simulator. And now the most relevant should be selected to solve the problem of predicting lane changes. The goal is to find the best set of features. The number of possible combinations is too large to find a solution with brute force methods. Therefore in a first step a pre-selection of the features is performed. Per hand just the features are selected that could reasonable have impact on the model performance. As soon as there are any doubts the feature is added to the pre-selected list. The list of 23 features can be found in the Appendix A.

But still there are too many features to try out all combinations. Therefore two different methods are used to make the search make efficient. As shown by Kudo and Sklansky the sequential floating forward search (SFFS) and the sequential floating backward search (SFBS) are good in performance and efficiency [27]. For the first step the state configuration is fixed and the model is rather simple, in our case it is one state for left and right lane changes and three states for staying straight. The assumption is that the best F1 score with a simple state configuration is also works well for more complex models. The two methods are explained in detail in the next parts.

SFFS

The idea of the sequential floating forward search algorithm is to build up a feature set sequentially. In the first step all available features are checked and the best is added to the selected feature pool. In the next iteration the combination of the selected feature pool and the rest of the available features are tested. The best combination is the new selected feature pool. This process is done as long as the new combination of features provides a better evaluation score than the combination before. This rather greedy way of adding features can have the disadvantage that a future combination may increase the score but not together with a strong feature added at the beginning. Therefore the floating step is included. After adding a new feature to the selected pool, the algorithm is removing every feature once and tests the score of the reduced combination of features. If the score increases the feature is removed permanent. The algorithm determines if both operations do not increase the evaluation score any more. A visualization of the idea is shown in Fig 5.1.

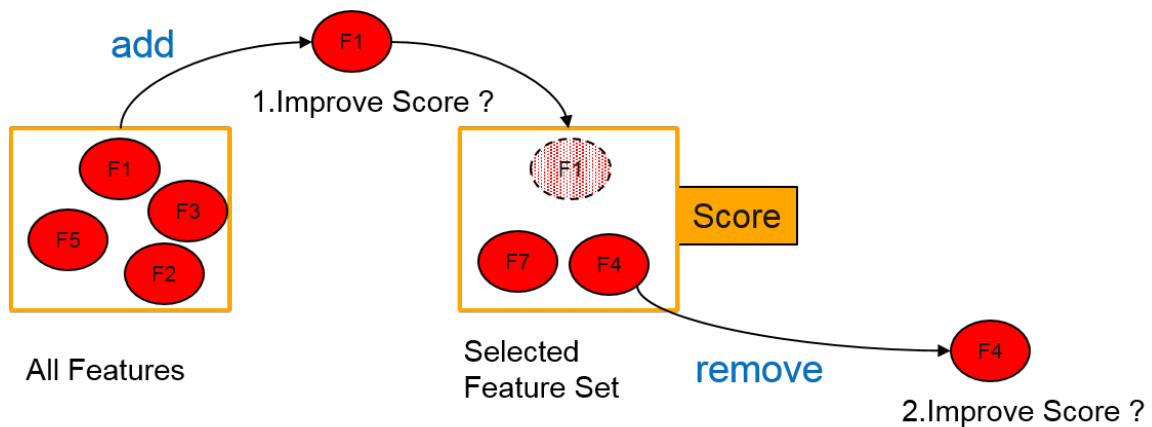


Figure 5.1.: Concept SFFS: First try to add the best fitting feature to the selected features, then test if a removal of one feature could increase the score.

The time complexity of the algorithm is $O(n) = 2^n$ where n is the number of features at the beginning [27].

In the implementation notice that it is also possible just to have one feature and the try to remove features can start if there are more than two features in the selected feature pool. Also, the attempt to delete the previous added element is unnecessary. According to [27] the SFFS Algorithm usually find good results on several types of datasets but it is still just a local optimum.

SFBS

The sequential floating backward search algorithm (SFBS) is similar to the SFFS described above. But this time all available features are in the selected feature pool from the beginning on. Then iteratively the features that decrease the score the most are removed until removing features does not increase the score any more. Additionally, the removed features are saved and after every iteration it is tried to add one of the features to the combination of features again. This algorithm is described in more detail in following pseudo code 5.2. Again the process determines if the evaluation score has been converged.

```

featurePool = allFeatures %with all Features
deletedFeatures = [] % empty at the beginning

bestScore =evaluate (featurePool)

while ScoreIsImproving

    foreach feature i in featurePool
        pool(i) = delete feature from complete feature pool
        Scores(i) =evaluate (pool);
    end

    check for best score in Scores
        move least useful feature to deletedFeatures
        save new bestScore

    %previous deleted feature could be useful in new combinations
    %so try to re add features from deletedFeatures and check
    % if the score is improving

    foreach feature in deletedFeatures
        add feature to pool
        scores = evaluate (pool)

        if scores improve bestScores
            add feature to featurePool
        end
    end

end
end

return featurePool

```

Figure 5.2.: Pseudo Code for SFBS implementation

Important in the feature selection methods is to split the data set in training, validation and test data.

5.1.3 Combine Feature and State Selection

The methods introduced so far assume that the feature and the state selection are independent. This would mean using more features does not influence the number of states. Because of the strong restricting another approach which takes this possible dependency into account is implemented. A simple brute force approach is used. For every combination of state configurations a new feature search is started. For feature selection either SFFS or SFBS are used. This process is very time consuming and therefore the search space is reduced. The search space is reduced according to the results of the feature and state selection with independence assumption (see Table 5.2).

But still there are 160 different state configurations and for every configuration a new feature search with complexity $O(2^n)$ with $n = 10$ has to be done.

Table 5.2.: Search space state selection

| | Lane Change Left | Stay Straight | Lane Change Right | Number of Features |
|--------------|------------------|---------------|-------------------|--------------------|
| Search Space | 1-4 | 2-10 | 1-4 | 10 |

5.2 On-line Learning Algorithms

In an on-line learning scenario the complete data is not available from the beginning. The data is streaming in, nevertheless the model should be able to make predictions all the time. Moreover, the storage for the data is not unlimited and less information as possible should be stored. This requirement fits with the limited computing and storage capacities of a car where more and more driving assistance systems are used. Using a cloud computing infrastructure would be a different scenario without this kind of limitations but this is not considered in this work.

5.2.1 Data and Labels

There are two kinds of on-line learning, one uses every single data point and another one that uses block-wise data (see [7]). In this work the block-wise case is preferred, because of the labelling process. The end point of a lane change is marked when the car is crossing the middle line. In order to find the start point of the lane changing intention, the algorithm is going back in time for a certain interval and search for a strong head movement signal. Therefore, a complete block of data has to be available to set the labels. After just seeing a single incoming data point it cannot be decided whether is point is part of a lane change manoeuvre because the manoeuvre may not be finished yet. So collecting a block of data is the first step then the labelling method is applied. Furthermore, the data blocks should be as small as possible due to the storage limitations. Because of working in the simulation environment these limitations are just theoretical but should be kept in mind.

The size of the data blocks can be changed for every scenario. The available data per driver is split into n parts so that every part has the same size. The data block sizes of different drivers might be slightly different because the overall time is not the same.

In practice the block sizes for training data range from three blocks a 20 minutes up to 21 blocks with a length about 3 minutes. Even smaller blocks are not desired in this setting because then too many data blocks do not contain any lane changes. All implemented algorithms ignore the data block if there is not a lane change to both sides. Staying straight is already the dominant state so no more training is needed for this part and a lot of blocks with just driving straight may also be not improving the assistance system. Other strategies may be possible like collecting data until there are enough lane changes and then start the training or train just the sub models separately even if not all three labels are available.

A higher frequency of updates can be archived when using a sliding window approach, here two or more small blocks are combined and used for training. In the next time step a new block arrives and replaces the oldest block. This needs the ability to save several blocks and the training algorithm can see blocks several times (see Figure 5.3).

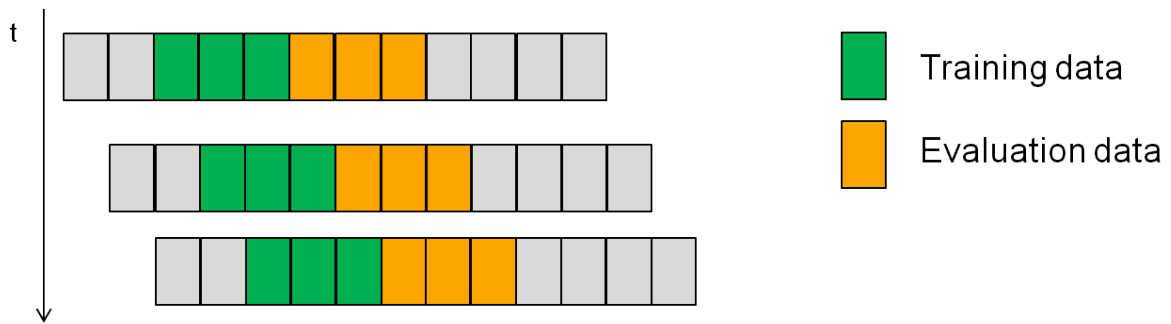


Figure 5.3.: Sliding Window: Development of the data blocks over time.

5.2.2 On-line Evaluation

An important characteristic for on-line learning approaches is that they can make prediction already during the learning process. Therefore also the evaluation of this kind of models has to be defined. The evaluation also needs the labels from the process step before and works on data blocks. Two ways of evaluation are possible. First the evaluation can be done on every single new data block. The current model is evaluated on the new data and then this block is used to update the model. The final F1 score is the mean value of all part-time scores. The second option is to use the same separate evaluation data set after every time step. In Figure 5.4 the difference is shown.

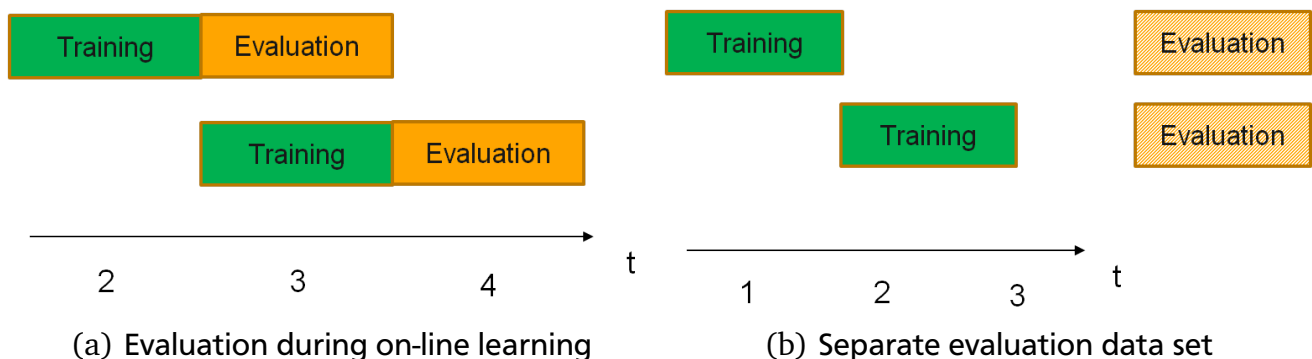


Figure 5.4.: Different evaluation strategies

The on-line learning should also adapt to temporal changes so calculating the F1 score after every data block, can represent also short term changes. Therefore the evaluation on the short incoming data blocks is preferred. In the sliding window approach just the last new recorded data block is used for evaluation propose.

5.2.3 On-line Algorithms

The algorithms from chapter 2.4.1 have been implemented so that they fit the needs of the lane change classification problem. Details are described in the next part.

Incremental Batch

One form of incremental learning is to use the normal EM algorithm for every data block. The process is iterating over the data and learns a local optimum for the model parameters. For the first data block the values are initialized randomly but for the following blocks the parameters from the previous learning session are used as starting values (see Figure 2.6). This approach should work better with larger data blocks and the first data block is critical because with a bad initialization the algorithm could get stuck in a local optimum for a longer time. As EM algorithm the same Baum-Welch form is used as in off-line learning.

HMM On-line Learning: Cappé

During this work it was not possible to get a stable and robust version of the on-line learning algorithm of Cappé. The algorithm is working in smaller examples but not with more than four states and a dimension higher than 4 for the emission probabilities. The Cappé algorithm uses a symbol-wise approach and tries to update to model like the EM algorithm but in an on-line way. In a simple example the algorithm is working like shown in Figure 5.5.

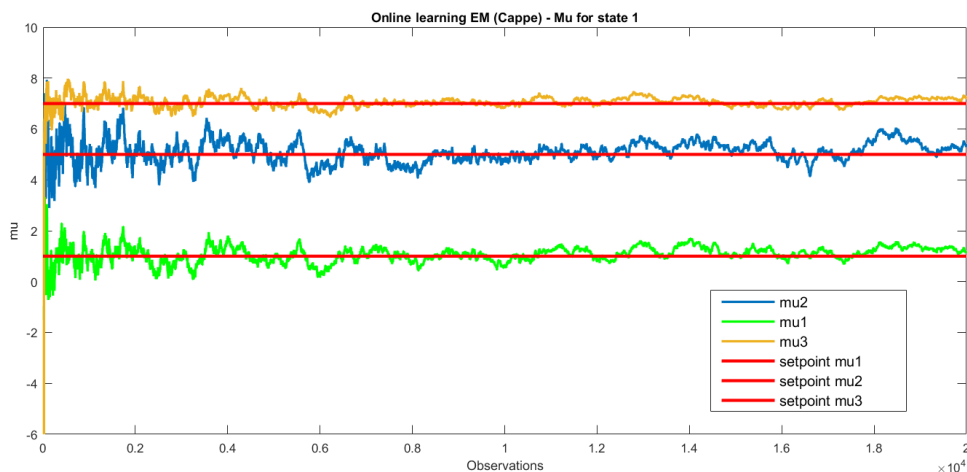


Figure 5.5.: On-line learning according to Cappé in a simple example: Mean values for the Gaussian emission probabilities are shown. From a known HMM the data is sampled and then used in training process.

Already for the covariance matrix the learned parameters are oscillating around the set value and no robust parameters are found. With higher number of states the matrix is not positive definite any more.

Badli's Algorithm

The algorithm from Baldi is implemented like described in section 2.4.1. To make the process more stable further steps are necessary. The main problem is that for the multi-variant Gaussian distribution a positive definite covariance matrix σ is needed. The update rule uses gradient descent techniques and cannot ensure that property. With a suitable learning rate the changes

in σ should be rather small and every new σ should be close to a positive definite matrix. After an idea from Nick Higham a method is used to find a positive definite matrix next to the given one. The idea uses alternating projections [19]. For Matlab exists a script which implements this behaviour ¹. Even if the corrected covariance matrix is not right the algorithm can continue and correct the failure later.

Pre-trained model

Instead of starting the on-line learning process with a random initialisation it is also possible to start with the general model as starting parameters. Whether this is a good idea depends on the driver. In the first case the driver is different to average just to a certain degree. So the general model is a good starting point and the on-line algorithm should be able to find this new optimum easily. But a general model fits, per definition, to the average, a new test person which differs a lot from the population which is used for the training the general model cannot use the general model as a starting point. In this second first case the on-line learning algorithm would have to forget about the general model parameters and then learn new individual parameters which are probably far away in the parameter space. The problem is illustrated in Fig 5.6

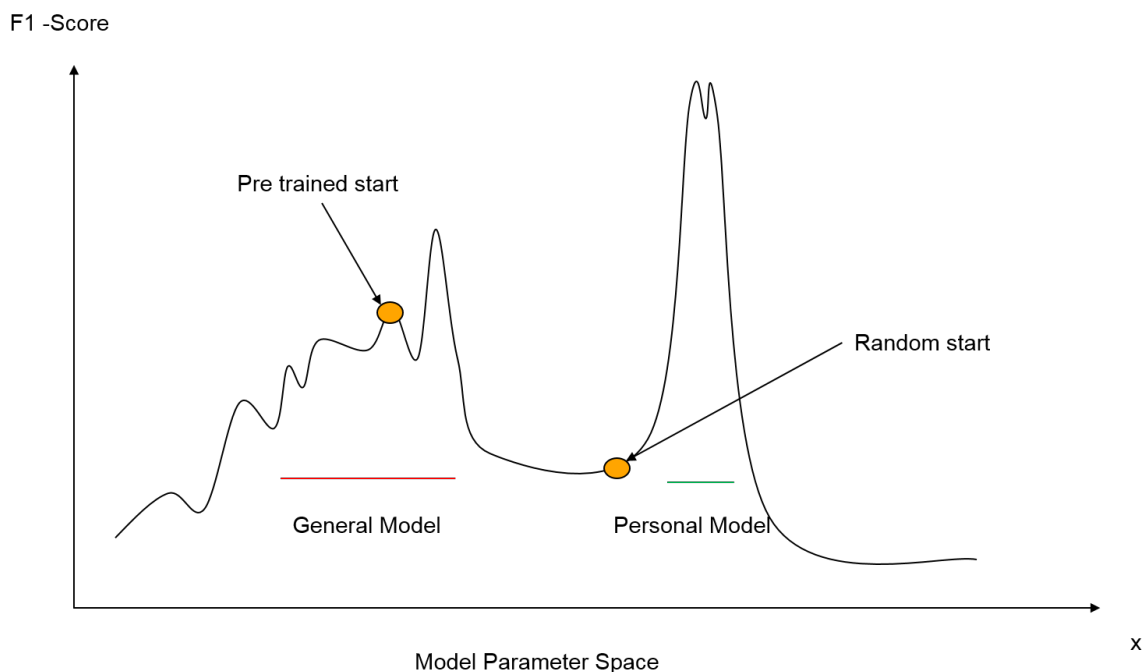


Figure 5.6.: Parameter Search: This figure should illustrate that for a driver with a parameter set far away from the average, it may be better to randomly start the parameter search instead of using the general model as a starting point.

It is hard to measure the differences in driving of the test persons in advance, especially the test population with six drivers is too small to evaluate these hypotheses. Nevertheless both approaches are implemented in the software.

The general model is learned on the data of the five other drivers, so that the general model has never seen any data of the driver itself. During implementation, it is important to save also

¹ <http://www.maths.manchester.ac.uk/~clucas/download.html>

the sub models and not just the final model.

When using the algorithm of Baldi it is necessary to find the inverse of the soft max parameters. This problem cannot be solved analytical but gradient method that tries to minimize the mean square error can find the parameters relative fast.

5.3 Model Management

A generic model management is implemented according to the graphics in 2.8. The idea is that keeping a pool of models could improve the overall performance and be more robust to the changes during the on-line learning. Because it is not sure that an on-line update always improves a model. In the current setting there are three different cases why the previous learned model do not provide good performance in the next step.

- corrupt data: the data is incomplete or the signals are disturbed, also not enough data may cause a problem.
- changes in the driver environment during the recording of the two following data blocks. For example for the next ride the driver is changing its seat position therefore all head and eye features may be shifted.
- changes in the driving situation: Even in the highway scenario there are slow and fast parts and less and more traffic this may affect the behaviour of the driver and then the prediction quality of the model. For example in a situation with low traffic the preparation of the lane change may be shorter and less careful.

Needed is the trade off between adapting fast enough to new situations and do not lose good model performance because of single disturbed data blocks. The model management described in 2.4.2 is a generic approach and could be used in many machine learning tasks. The pseudo code shown in 5.7 and the following parts explain how to use it in the case of classification of lane change intentions. First important part is the model pool and how to initialize it. Furthermore the process of updating the models and selecting the models to the final prediction is described. Therefore it is necessary to think about how to manage the data and at what time use a certain data block. The model pool can potential grow to infinite therefore it is necessary to delete certain models that do not produce good results for a longer time interval. Next to the training process the prediction method also can be improved and in the ideal case the best models are used to predict the manoeuvres.

```

%load config
configmgmt = loadConfig(configname);

%data pre processing and splitting
[data, label] = dataProcessing (configmgmt);

%create modelPool
modelPool = createModelPool (states, feature, updateMethods);

for every new data block as evalData
%special case first data block
    if(counter==1)
        [modelPool, selectForVote] = firstTryManagement (modelPool, data);

    else
        evalData = getEvalData();
        trainData = getTrainData();
        data = [evalData, trainData, longEvalData];
        %prediction and evaluation
        [learnedManeuver] = prediction (selectForVote, data);

        %majority Vote - mVote - current eval Set
        mVote = evaluation (learnedManeuver, evalLabel);

        %model managment with update and selection
        [selectForVote, modelPool] = activeModelManagement (modelPool, data);
    end
end
end

```

Figure 5.7.: Pseudo Code Model Management

5.3.1 Model Pool

The model pool is a data structure that holding several models. First it is defined what is a model, then what kind of characteristic are needed in our case and then the next parts answering the question of how to initialise the model pool.

Models

The models inside the pool should be different, there is no gain of having a model pool with a lot of very similar models. Therefore the idea of model classes is introduced. Model classes in this context are not completely identical with classes that are known from the object oriented programming paradigm. But it uses some key concepts. So from every class there are multiple instances. An instance of a model class has same the model configuration but the model parameter may be different. So an instance shows a model in different learning state. Furthermore the model class also defines additional properties of the model, like the current evaluation score.

In the case of Hidden Markov Models a model class is defined by its state configuration and its feature set (see Fig 5.8). Moreover, following properties are used.

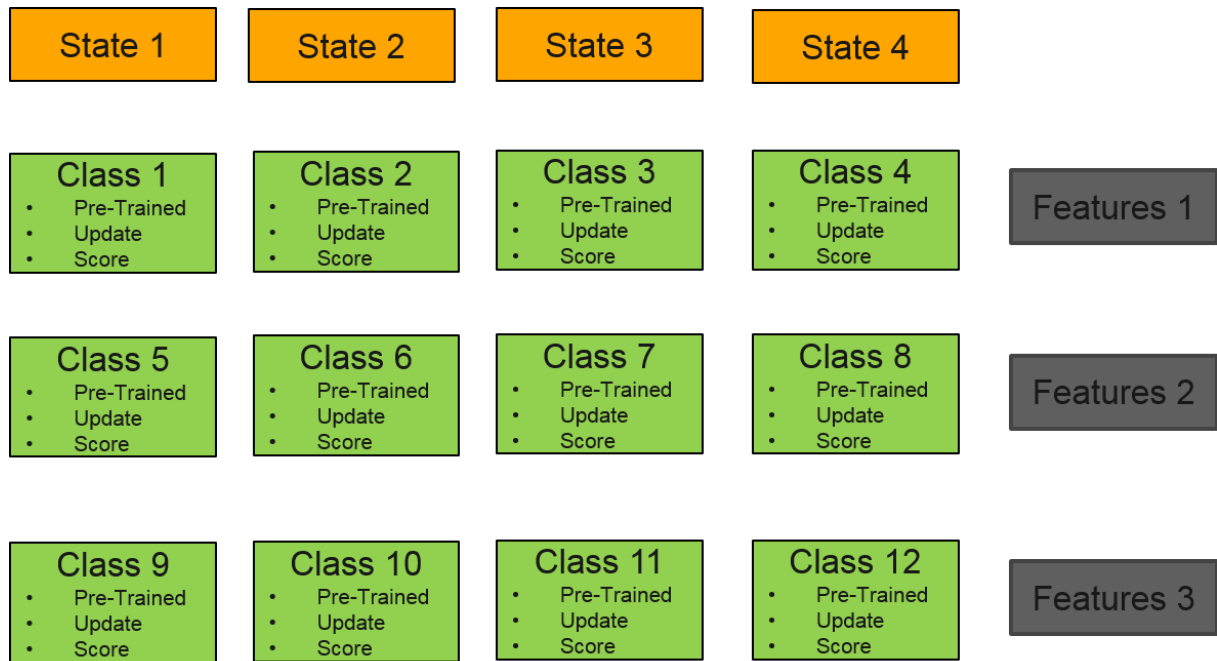


Figure 5.8.: Combination of model classes, each class is defined by its feature and state configuration

- *id* is a number with 3 digits, first digit indicates the feature set, second one the state configuration and the last digit encodes the update method.
- *configName* is a string that describes feature and state configuration.
- *updateMethods* gives a string that stands for a certain update method. A model class can have different instances with different update methods. In the long run one update methods should dominant all others.
- *stateConfig* is an array with the number of hidden states for the sub HMMs left, straight and right.
- *featureSet*: a string array that contains all feature that should be used for this model class. The names have to match with the row names in the data set.
- *preTrained* described if a pre-trained model is used to initialized the model in this class. At the moment this is just a boolean value and the pre-trained model is loaded during runtime according to a naming convention using the *configName* property.
- *modelData* is the container for the model itself. In our case it is a Hidden Markov model and its parameters. For the on-line learning process it is important that the model also contains all sub models for the different classification classes.
- *maxInstances* limits the maximal number of instances for the model class. This makes sure that there not too many instances and the model pool is growing to infinity.
- *Time to live (TTL)* indicates the long term performance of a model class. A model class could be provide low performance with a longer time horizon. So the TTL counter is reduced if the model performs under average.

-
- *score* shows the score of the model instance.

Create Model Pool

In one of the first steps the model pool has to be initialized. The set of features and the number of states per manoeuvre are different per driver. The combinations of a pre-selected set of different features sets and several distinct state configurations are used to define the model classes. Furthermore for every class several instances are created at the beginning. The method that creates the model pool also uses an array with the implemented update methods as input. So for every model class an instance for each update type is created. Also for the scenario with a pre-trained model a separate instance could be added to the model pool. This is necessary to get different starting points, after a longer run a certain update strategy should dominate the other update methods and the pre-trained and the random initialized model should either find the same (global) optimum or if stuck in different local optimum. But every model class is initialized with at least one instance at the beginning.

The further model class properties are set according the configuration of the experiment or to the default values. The score is initialized with zero, TTL and maxInstances are equal for all instances. If a pre-trained model should be used, it can be indicated with the name of the update method. A function is checking for the prefix 'pre' and handling this input. As an update function it is also possible to define 'none'. This means the model is not changed at all and functions like a static model, this just works out with pre-trained models. So a general model can be added to the pool and this could be the fall-back solution (if the TTL is set high enough).

Following model classes are implemented in the test scenarios in this work. The selection has been made according to the evaluation in the off-line learning case and the classes should be distinguishable from each other.

- Feature sets
 - $F1 = \{ 'LateralDistance', 'psi', 'HeadHeading' \}$
 - $F2 = \{ 'HeadHeading', 'LateralDistance', 'psi', 'Steering Torque', 'dLongSameAheadNext', 'yaw', 'SteeringWheel' \}$
 - $F3 = \{ 'Steering Angle', 'Steering Torque', 'LateralDistance', 'psi', 'pitch', 'roll', 'dLongSameAheadNext', 'HeadHeading', 'HeadPositionX' \}$
- State configuration
 - State 1 = [1 3 1]
 - State 2 = [1 7 1]
 - State 3 = [1 4 2]
 - State 4 = [2 4 1]

All model classes are available with either random initialisation or with a pre-trained model. This means for each driver, 12 pre-trained models according to the model classes are necessary.

As update methods incremental batch learning (*batch*) and the on-line learning after Baldi (*baldi*) are available. Moreover there is also the option not to update the model (*none*).

Due to implementation issues the each model class have its own sub model pool. The complete model pool is the set of all these sub pools.

5.3.2 Data Management

Because the live data stream is just simulated, during the implementation the order of the data has to be handled carefully. To follow good machine learning practice the evaluation data should not be used for training purpose before. Nevertheless the data should be used as efficient as possible.

Unless the model is pre-trained in advance during the first data block no prediction can be made. From the next step, on every incoming data block is used for evaluation in the training process and of course the model is predicting after every data point. The previous data block is used for training and updating the models. The concept can be seen in Fig 5.9.

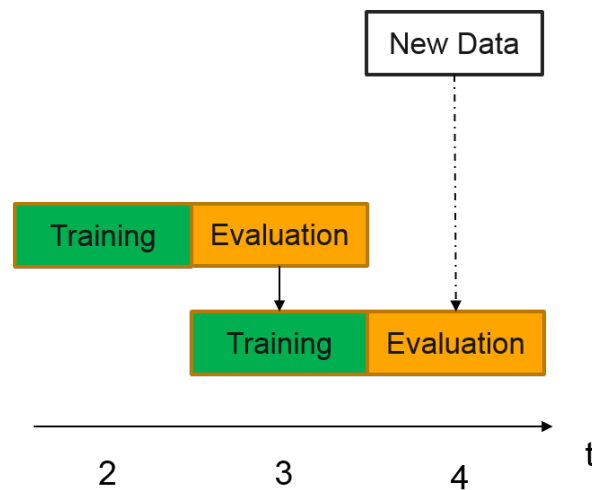


Figure 5.9.: Data management for the Model Management, new data is first used for testing the models and then for training in the next time step.

Different feature sets also mean that not always all parts of the data block is needed. So before every training or evaluation step the data has to be filtered according to the feature set of the current model class. The basic data set contains all 23 features that are chosen in section 5.1.2.

5.3.3 Update function

After receiving a complete data block the model pool have to be updated. The *updateModelPool* function is iterating over all model instances in the model pool. First for every model the given training and evaluation data has to be filtered according to the feature set of the model class. In the next step the update method which is saved as model property is called and the model parameters are updated. In the next step the new model is evaluated on the separate evaluation data set. That score shows what performance the model would have on the current data block or the external evaluation set.

A temporary decrease in the model performance is possible due to changes or local optima. So if after an update the model performance is lower than before, the model is still valid and can be developed to a stronger model over time. In such a case the old and better model is saved and kept in the model pool, because model quality is improved. But also the new model with the worse performance is cloned a saved in the model pool with the new parameters. The

selection before the prediction part can be done either on the short evaluation data set or on the testing data set D4. For the short term models a new model needs a better score to be saved. The process is shown in Fig 5.10.

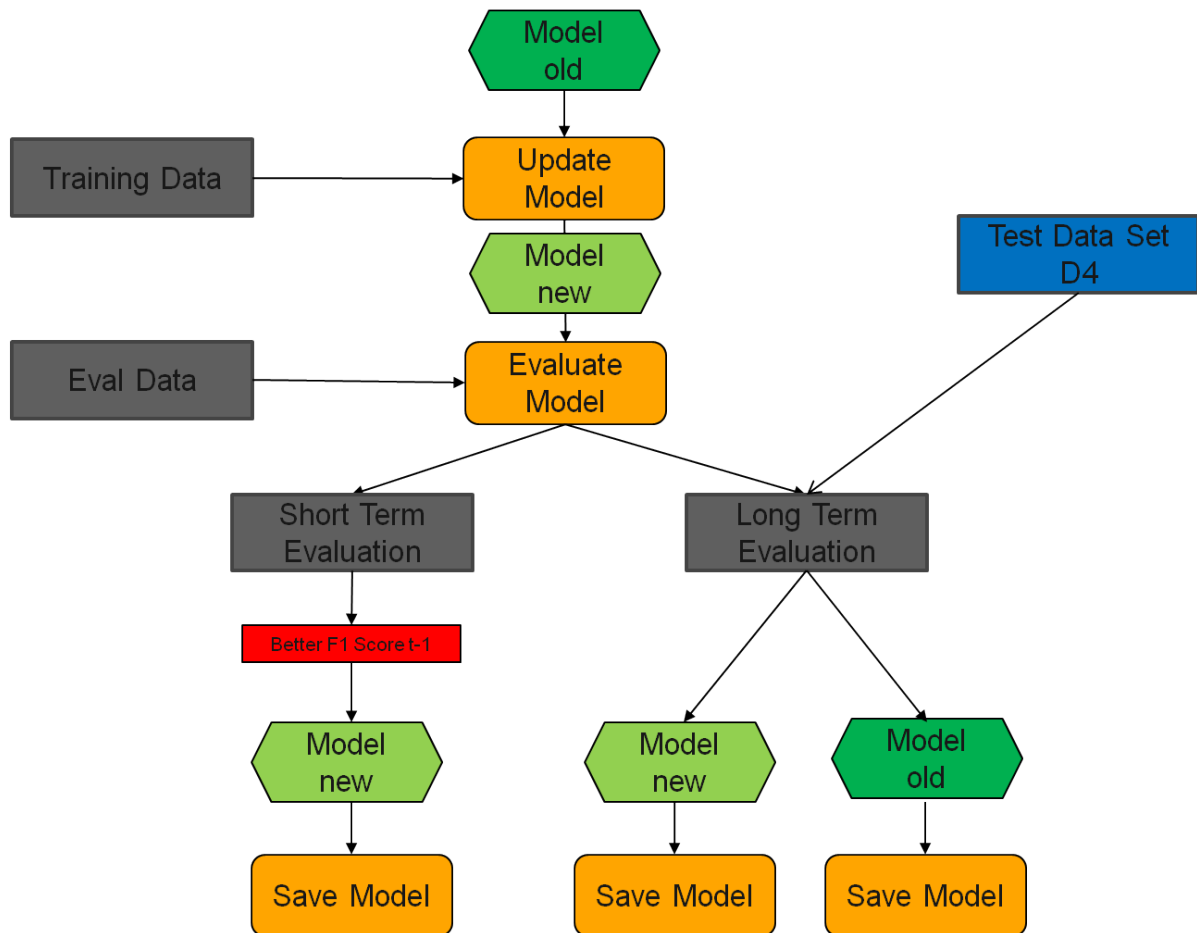


Figure 5.10.: Update process for the model management: model score and model parameters are updated.

This kind of implementation allows the model pool to grow very fast and this is storage consuming and the computational time is also rising quickly. Therefore some methods are used to limit the growth of the model pool, they are described in the next section.

5.3.4 Model Pool Management

A first measure is to limit the maximum number of instances per class. This deals with the problem that after every model update also the weaker candidate is saved and added to the model pool. With the *maxInstance* attribute it is checked that if there is a free spot in the sub model pool. The sub model pool is defined by the class the instance belongs to. If there is no free space available the model that is not used for prediction the longest time is deleted. And the new model is added to the pool.

As an advantage could be named that it limits the model pool size and very poor models are not kept in the model pool. But it also has got the risk that if the *maxInstance* parameter is too small the variety is getting lost too fast. In a long term view the distinction between pre-trained

models and different update strategies will vanish.

Not just model instance could perform below average, but also model classes. In order to reduce computation time, these classes are deleted after a longer time period. Probably every driver fits best to one or more model classes, but it is not known in advance. The *TTL* property of the model class is a variable that handles the time how long low performing classes are hold in the model pool. It is reduced if the best instance of a model class performs under average. As an alternative also the median value could be used as criteria. To slow down this process it is also possible to increase the TTL counter if the model is selected as an over average model class.

5.3.5 Prediction and Model Selection

Next to the training process also the live prediction has to be handled. Having a pool of models gives some new opportunities how to be successful in this field.

Single Vote

The straight forward idea is to select the best model form the model pool and use it for prediction in the next time step. The selection function simply becomes a maximum operator over all models. The models are evaluated during the update process, so the performance on the previous data block decides which model is used for prediction. This method is simple and easy to implement but also has disadvantages. Just using one model is endangered to over fitting even if the time intervals are rather small.

To make even more use of the model pool the concept of majority vote is implemented and explained in the next part.

Majority Vote

The idea is to use several models and let them vote which manoeuvre they predict. Every model does its prediction on its own and in the end all votes are counted and the prediction with the most votes is given to user. If all models are taking part at this election also the rather poor performing models have a lot of influence on the decision. Therefore a pre-selection is necessary and the function *selectedModels* is called. The naive idea is just to select the best N models has the problem that probably a lot of very similar models are chosen and there is not a big difference to just taking the best on. Therefore in our implementation the best model of each model class is calculated. So per design these models can be distinct by feature and state configuration. To not use inappropriate model classes just model classes with a score higher than average are used to make the final prediction.

For the voting process there are two different options available. In the first approach every selected model does its prediction and the state with the most votes wins. In the second scenario a weighted majority vote is implemented. So every vote is weighted with some factor. This factor is the current score of the model. So models with a good performance, in the last data block are more important in the next prediction cycle.

Select models

Three different methods for the model selection are available. First the evaluation on the last data set is used as metric to decide which models are chosen for the final prediction. So the models can adapt fast to temporary changes. In the long term view this model may not be a good generalization and corrupt data may destroy the old well performing models. So the second approach is to evaluate all models on the test data set D4. The third approach extends the current model management to a extra model pool just for the short term models. So the normal model pool is evaluated on the long term data set. Moreover all models are also tested against the short term data block and the best model for each model class is saved. In the prediction mode all models are used, after passing a filter (above average score or a fixed threshold). So on the long term view the model can adapt to the driver's behaviour and also short time changes are recognized.

Sort Data

Every test run contains four different highway sections which are repeated twice and every test person has to complete the course twice as well. So four times every highway section is passed. Because there is the hypothesis that the type of highway and traffic situation may influence the driver's behaviour it is useful to re-sort the data so that similar sections are connected. That should improve the performance of the on-line algorithm because with more data it is more time to adapt to the different sections. The sorting in the simulation environment is easy, every highway section has its own unique id. The disadvantage of the method is that there are jumps in the data when the data parts are reconnected.

Evaluation

The model is evaluated on also on the current data set. The total performance of the model per driver is the mean of all the F1 scores on every data block. This is the most realistic performance metric. But the evaluation blocks are relative small so as an alternative method the score could also be compared with the test data set D4 at every time step.

As a baseline score a general model is chosen for every driver, model configuration is personalized. Then this model is evaluated on every small data block.

6 Evaluation and Results

The evaluation chapter is divided into three parts. First topic is personalization, this mean how does the model parameters influence the performance and to what extend the structure of the model is individual. These considerations are made in an off-line scenario, so all data is available from the start on. In the second part is dealing with the performance of the on-line algorithms. Finally, these incremental learning algorithms are integrated into a more complex model management. In this ensemble learning setting several parameters can be optimised.

6.1 Personalization

This section tries to find an answer to the question, whether it is useful to learn personalized models instead of using a generalized model and how much the performance can be improved. First of all the data handling and split is explained.

6.1.1 Data

The complete data set per driver is split into four parts (see Figure 6.1). Each part contains all different course sections. The last data set D4 is always used for evaluation and testing. The first parts are for training.

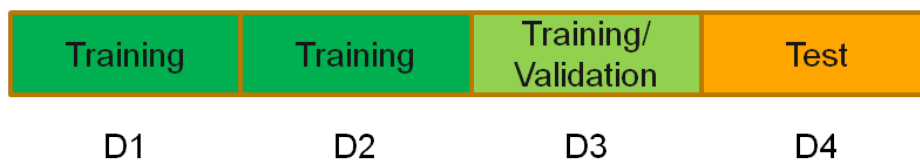


Figure 6.1.: Data is split into four parts, each block of data is about 20 minutes long.

The part D3 is either used for training also or as a validation set if hyper parameter of the model have to be chosen.

6.1.2 Standard Model Learning

In the previous work of Faller [16] a model is described that performs best for a generalized model. The data and setting is very similar, therefore this type of model is used to run some first tests. The model consists of a state configuration 1-7-1 which means the sub HMM for lane change left and right is modelled with just one state. With seven hidden states the sub model "staying straight" is represented. As a basic feature set the following seven features are the result of the feature selection in this work.

- 'HeadHeading': head movement relative to the axes
- 'LateralDistance': distance to the middle line

- 'psi': difference between heading angle and lane tangent
- 'Steering Torque': steering wheel torque
- 'dLongSameAheadNext': Distance to the next car ahead on the same lane
- 'vyaw': speed of the heading angle
- 'SteeringWheel': angle of the steering wheel

This model is called standard model from now on.

General Model

With the configuration of the standard model a general model is learnt. For every driver the model uses all the data of the other drivers. So a leave one out cross validation is performed. The result is shown in Fig 6.2.

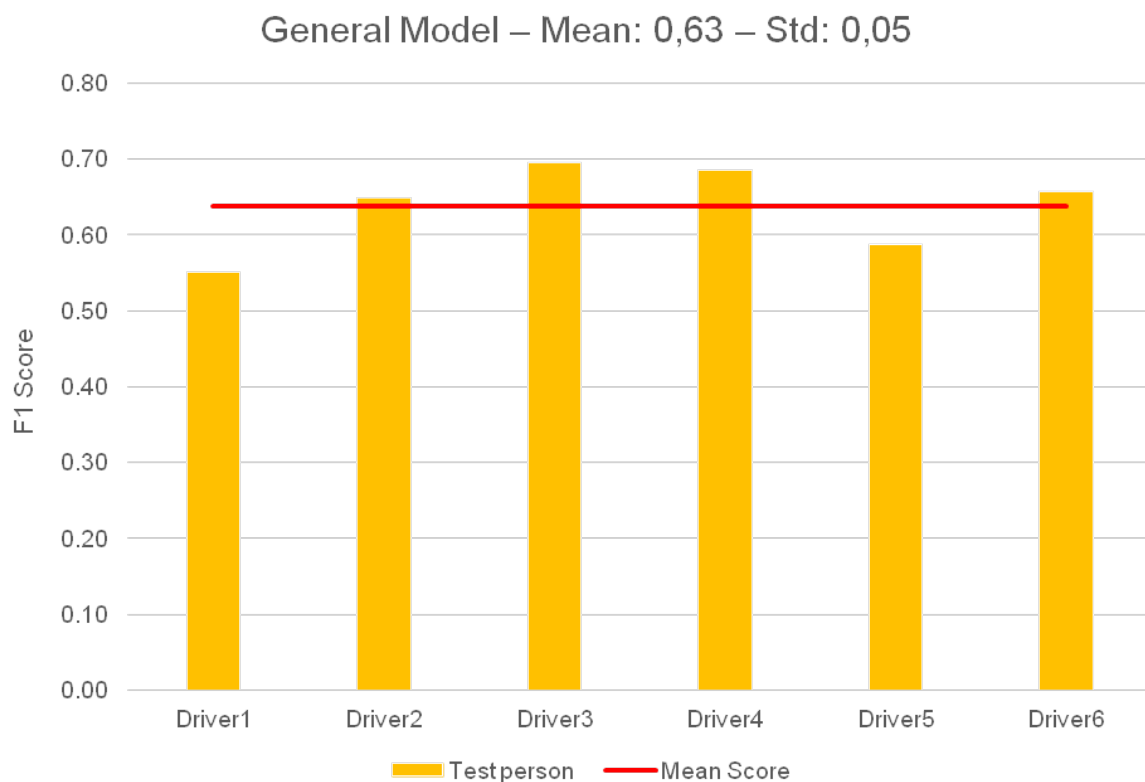


Figure 6.2.: For each driver a general model is learnt on the data of the five other drivers and then evaluated.

With just six different drivers the generalization of the model is rather poor. More data of different kinds of driver would improve the results, but this was not a key interested of this work. So this F1 score is just for basic orientation.

Individual Model

In the next experiment the same model configuration as above is used. But the hypothesis is that the behaviour of every driver is significant different. For example the extent of the head movement when looking back before a lane changing manoeuvre is different. Therefore, in the next step for every driver the model is trained on just the training data sets D1 to D3. So the model can adapt to the special driving style of the test person.

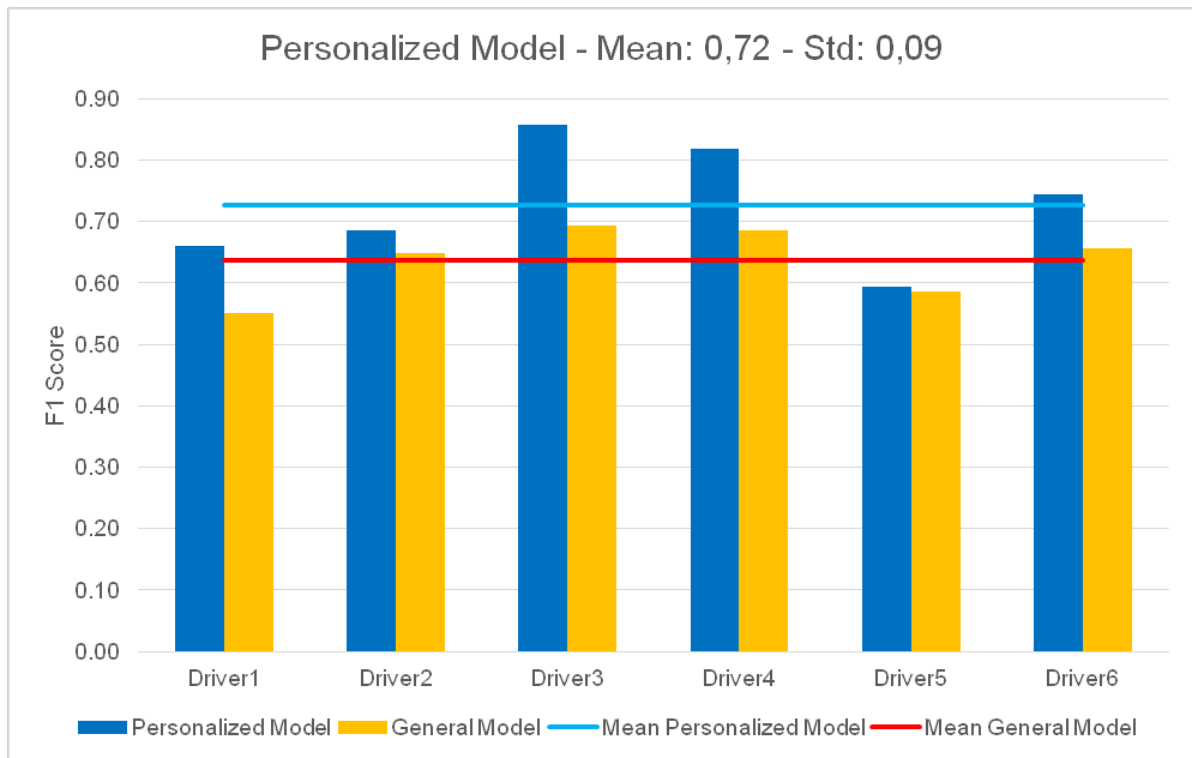


Figure 6.3.: Personalized Models compared to the general Model

The average F1 score of 0,7268 is about 0,1 points better than the general model. So the personalized approach is necessary and the total amount of data is much lower than for the general model. So the data quality matters a lot.

All drivers could improve the score compared to the global model. But the difference depends a lot on the driver. So the standard derivation is doubled as high as before. So the idea that the model configuration is not suitable for all drivers comes up. Therefore in the next section a selection method is evaluated to find the best model configuration considering states and features.

6.1.3 Feature Selection

In a first step the complete list of 300 features is reduced to a set of 23 features. So that just features are selected that may affect the lane change intention. All features that give background information about the course which are not available in a real world scenario are not used. According to Doshi and Trivedi the head movement is more important and more robust than the gaze direction features [15]. The resulting selection contains just continuous features, although

features like the discrete world intersections could influence the lane change manoeuvre. But first this feature should correlate with the head and eye movements and also in [16] it has been shown that this feature has got low influence on the model performance. The complete list can be found in the appendix A.1.

SFBS

To find the best set of features for each individual driver further search methods are necessary. The search space is still too large so the Sequential Floating Backward Search (SFBS) and the Sequential Floating Forward Search (SFFS) are implemented. For both search algorithms the data is split into following parts. D4 is still the test data set. The model is trained on D1 (or D1 and D2) and then a validation is made on D3. According to the score on the D3 data set the best model configuration is chosen.

Concerning the F1 score these are the interesting results:

- The mean F1 score over all driver on the test set D4 is slightly lower than the score on the standard feature set. But on the test set D3 the score is higher and the standard deviation is also higher (0,09 compared to 0,12). Most of the models for the drivers are improving or stay the same but for some driver there is a large decrease in performance. This indicates that the search algorithm get stuck in a local optimum.
- The score on the validation data set D3 reaches 0,53 as F1 score after just selecting the first feature ('Lateral Distance' or 'HeadHeading'). This shows that these both features are very dominant.
- Adding additional training data has no positive effect in this experiment.

The Figure 6.4 shows which features are really selected. From starting with 23 features the algorithm terminates in average with a set of 19 features. So just three or four features are deleted. The average score is similar to the score which have been reached with the standard model. The standard model has nine seven features. So it seems that there are several features that do not improve the model performance but also they do not distract the prediction.

| | Feature | Driver1 | Driver2 | Driver3 | Driver4 | Driver5 | Driver6 |
|----|----------------------|---------|---------|---------|---------|---------|---------|
| 1 | AcceleratorPedal | x | x | x | | x | x |
| 2 | Steering Angle | x | x | x | x | x | x |
| 3 | Steering Torque | x | x | x | x | x | x |
| 4 | LateralDistance | x | x | x | x | x | x |
| 5 | psi | x | x | x | x | x | x |
| 6 | ax | x | x | x | x | x | |
| 7 | ay | x | x | x | x | x | x |
| 8 | pitch | x | x | x | | x | x |
| 9 | roll | x | x | | x | x | |
| 10 | v | x | x | | x | x | |
| 11 | vyaw | x | x | x | x | x | x |
| 12 | yaw | | | | | | |
| 13 | dLongSameAheadNext | x | x | x | x | x | x |
| 14 | dLongSameBehindNext | | x | x | x | x | x |
| 15 | dLatSameBehindNext | x | x | | | x | |
| 16 | HeadHeading | x | x | x | x | x | |
| 17 | HeadPitch | x | x | x | x | x | x |
| 18 | HeadRoll | x | x | x | x | | x |
| 19 | GazePitch | x | x | x | x | x | x |
| 20 | EstimatedGazeHeading | x | x | x | x | x | x |
| 21 | HeadPositionX | x | | x | x | x | x |
| 22 | HeadPositionY | x | x | x | x | x | x |
| 23 | HeadPositionZ | x | x | x | x | x | x |

Figure 6.4.: Matrix of selected features through SFBS - in grey the features from the standard model are marked

The model complexity increase with more features, so in a next step the forward type of the algorithm is used to find a more compact feature set.

SFFS

In the SFFS version of the search algorithm, the starting selection set is empty and it is filled up over time. The overall performance can be improved compared with the standard model. As shown in Fig 6.5 especially the scores of drivers with a lower score before improve.

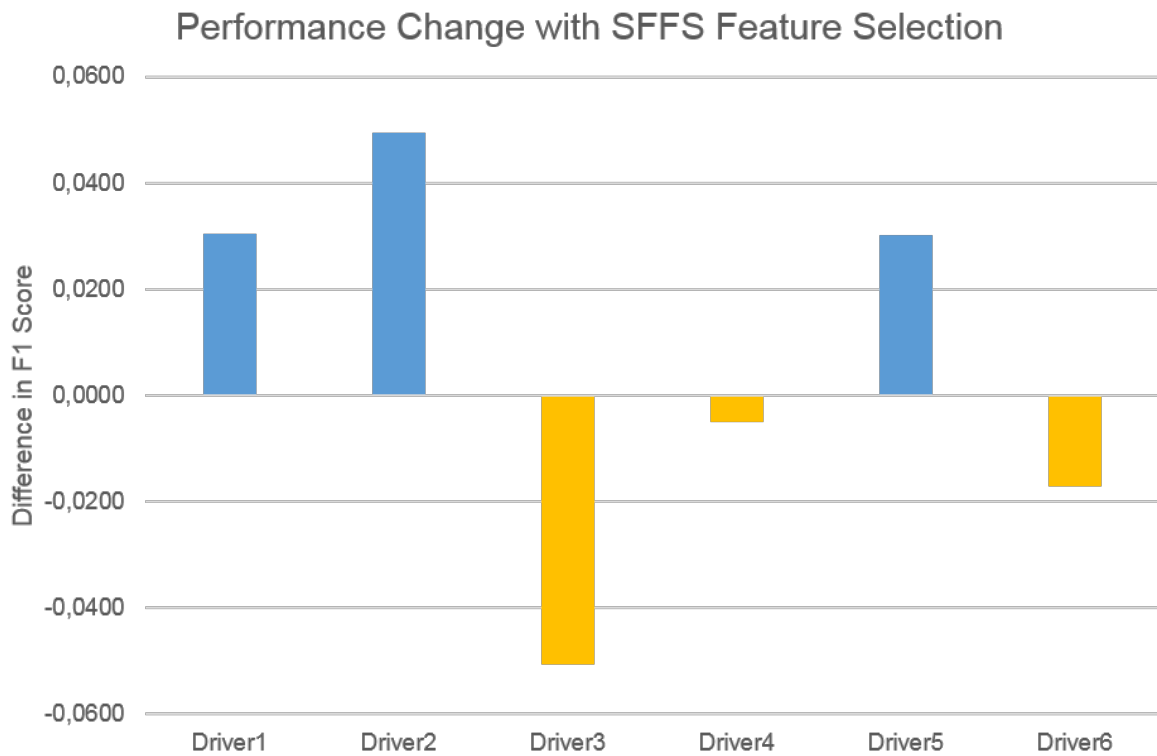


Figure 6.5.: Performance development after SFFS feature selection compared to the standard model

In opposite to the SFBS algorithm the results for SFFS are better with both training data sets. The resulting feature sets are very different. With the SFFS method in average six to seven features are selected compared to about 19 in the other case. An overview of the selected features is depicted in Fig 6.6.

| | | Driver1 | Driver2 | Driver3 | Driver4 | Driver5 | Driver6 |
|----|----------------------|---------|---------|---------|---------|---------|---------|
| 1 | AcceleratorPedal | x | | | | | |
| 2 | Steering Angle | | x | x | x | | |
| 3 | Steering W Torque | x | | | x | x | |
| 4 | LateralDistance | x | x | x | x | x | x |
| 5 | psi | x | x | x | x | x | x |
| 6 | ax | x | | | | | |
| 7 | ay | | x | | | x | |
| 8 | pitch | | | x | | x | |
| 9 | roll | | | | x | x | x |
| 10 | v | | | x | | | |
| 11 | vyaw | | x | | | | |
| 12 | yaw | | | | | | |
| 13 | dLongSameAheadNext | | | | x | | |
| 14 | dLongSameBehindNext | | | | | | |
| 15 | dLatSameBehindNext | | | | | | |
| 16 | HeadHeading | x | x | x | x | x | |
| 17 | HeadPitch | | | | | x | |
| 18 | HeadRoll | | | | | | x |
| 19 | GazePitch | | | | | | |
| 20 | EstimatedGazeHeading | | | | x | | |
| 21 | HeadPositionX | | x | | | | |
| 22 | HeadPositionY | x | | | | | |
| 23 | HeadPositionZ | | | | | | |

Figure 6.6.: Matrix of selected features through SFFS. In grey the features from the standard model are marked.

In the table 6.1 the features that are most often selected are shown. These statistics are made over the example of just one data block of training and the case with training on D1 and D2.

Table 6.1.: Most frequently chosen Features

| Feature Name | Selection Rate |
|----------------------|------------------|
| 'LateralDistance' | 100,0 % |
| 'psi' | 100,0 % |
| 'HeadHeading' | 83,3 % |
| 'roll' | 50,0 % |
| 'Steering Torque' | 41,6 % |
| 'vyaw' | 33,3 % |
| 'dLongSameAheadNext' | 33,3 % |
| 'pitch' | 33,3 % |
| rest | less than 30,0 % |

In conclusion to both feature selection algorithms, it is to say that on the data set of this work the forward version works better than the backward approach. But for some drivers both versions stop in local optima and do not reach the performance of the standard model. The SFFS methods tends to find a feature set that is smaller than the SFBS, this leads to a less complex models and also shows better generalization on the evaluation data set. The fact that lower

performing test person are improved shows that the feature set of the standard model is a good generalization and it can be improved just in certain scenarios .

So far the experiments have been done with a fixed state configuration. In the next section the number of states is changed.

6.1.4 State Selection

The standard model is parametrized with a state configuration 1-7-1. So changing the lane is just modelled with one state while stay straight needs 7 states, which is probably because the variance of driving straight over many drivers is represented in these seven states. For the personalized learning scenario one hypothesis is that the optimal number of states for staying on the lane would be decrease.

To find the optimal state configuration a simple brute force search is implemented. The search space is limited to 1-5 for the changing right or left and 1-15 states for stay on the current lane. The data is split into training (D1 and D2), validation (D3) and test (D4) data sets. The feature selection is fixed and is the same as the standard model.

Results for State configuration per driver

As a result of the test runs described above a state configuration per driver is found and shown in Table 6.2. The overall mean F1-score on the evaluation data set is 72,81 %, which is similar to the performance of the standard model. The decision is made on the validation data set here a performance score of 79,28. So also on the state configuration an over-fitting is possible.

Table 6.2.: State Selection

| Driver | 1 | 2 | 3 | 4 | 5 | 6 |
|---------------------------------------|--------|--------|-------|--------|-------|-------|
| State configuration (Right-Stay-Left) | 4-10-2 | 1-13-1 | 1-4-2 | 3-10-4 | 2-6-1 | 1-4-2 |

The hypothesis with the fewer states for the stay state is just true for half of the drivers. Also mixed is the behaviour of change left and right some test persons are modelled with more states on the right some need more states to represent the manoeuvre to change left.

Reduce Model Complexity

With more than 11 states the model complexity is higher than the standard model (9 states) but the mean F1 score is not improving. So there is some kind of over-fitting involved. To find a good trade off between model complexity and performance the formula 5.1 is used to weight the performance according to the number of used states. As a result Figure 6.7 shows this new metric.

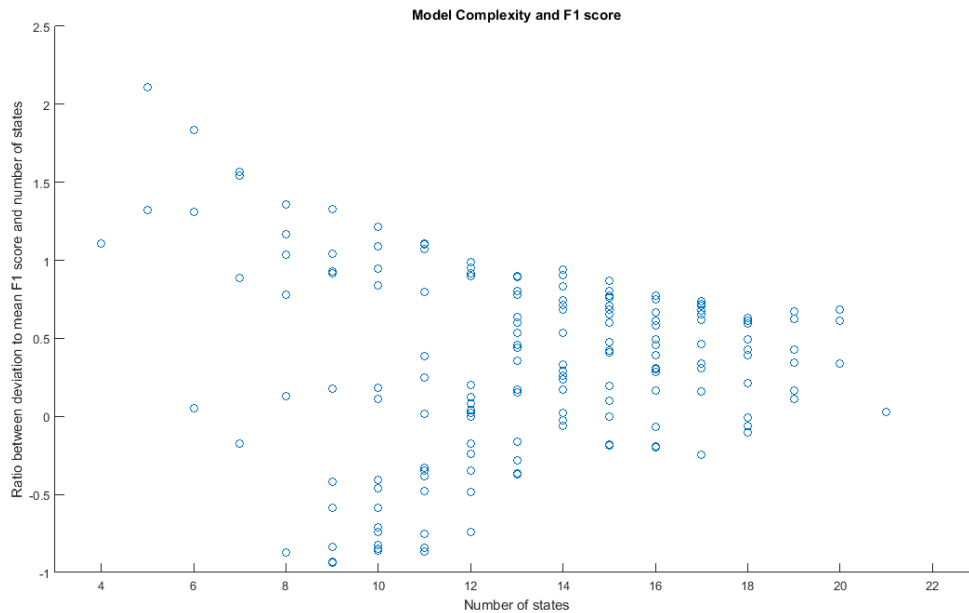


Figure 6.7.: Ratio between deviation to average F1 score and number of states - higher values indicate a good performance with minimum number of states.

Using this metric to decide about the optimal state configuration leads to new state configurations per driver, Table 6.3.

Table 6.3.: State Selection with reduced complexity

| Driver | 1 | 2 | 3 | 4 | 5 | 6 |
|---------------------------------------|-------|-------|-------|-------|-------|-------|
| State configuration (Right-Stay-Left) | 1-3-1 | 1-3-1 | 1-4-1 | 1-3-2 | 2-4-1 | 1-4-2 |

Now in average just six states are needed and especially the states for stay on the lane are reduced. Although the average F1 score is with 71,32 % a little lower than with the method before, the result shows a good trade off between performance and model complexity.

6.1.5 Combined Feature and State Selection

The findings of feature and state selection are combined. So the model is trained with the best feature set and the optimal state configuration for each driver. The evaluation on data set D4 shows that the performance of the models does not increase in average. The F1 score is similar to the standard model, although single driver can gain better scores. But other drivers stuck in some local optimum which indicates that probably the amount data for making this decision was not enough.

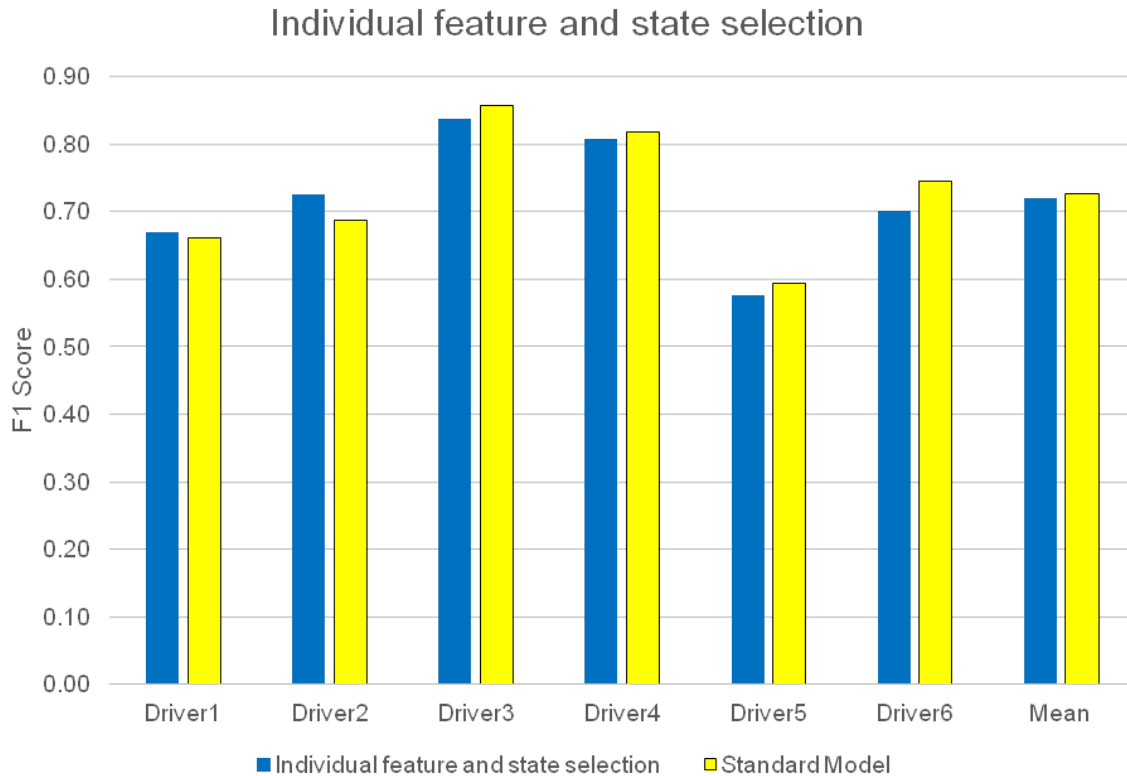


Figure 6.8.: Combine the results from feature and state selection

So far feature and state selection have been considered separately. The assumption behind this approach is that the feature selection is independent from the state selection. This seems not complete true because the combination of feature and state selection shows similar performance scores. Therefore a combined search over feature and state space is realised. To do this in reasonable time the state search space is reduced and also the features are limited to a set of 10 entries. The reduction is made according to the findings of the previous step.

For every state configuration (left and right 1-4 state and straight 2-10 states), a SFFS feature selection is performed. The results do not show any improvement in the average F1 scores to the methods before. Maybe the cause of the limited training data and the reduced parameter space, also this higher dimensional search space may produce more local optima. Compared to the standard model, single driver could improve, but the enhancement is similar to the methods which uses state and feature selection separately.

6.1.6 Summary and comparison

The last sections show that it is useful to train the models in an individual way. Choosing the best model for personalized learning for every driver, an average F1 score from 75,25 % can be reached. Table 6.4 shows the score per driver and its model configuration.

Table 6.4.: Best Performance - Personalized Off-line Training

| | F1 Scire | State Configuration (L-S-R) | Feature Set |
|----------|----------|-----------------------------|---|
| Driver 1 | 0,6910 | 1-7-1 | 'AcceleratorPedal', 'Steering Torque', 'LateralDistance', 'psi', 'ax', 'HeadHeading', 'HeadPositionY' |
| Driver 2 | 0,7374 | 1-3-1 | 'Steering Angle', 'Steering Torque', 'LateralDistance', 'psi', 'pitch', 'roll', 'vyaw', 'dLongSameAheadNext', 'HeadHeading' |
| Driver 3 | 0,8569 | 1-7-1 | 'LateralDistance', 'psi', 'Steering Torque', 'dLongSameAheadNext', 'vyaw', 'Steering-Wheel' |
| Driver 4 | 0,8180 | 1-7-1 | 'LateralDistance', 'psi', 'Steering Torque', 'dLongSameAheadNext', 'vyaw', 'Steering-Wheel' |
| Driver 5 | 0,6239 | 1-7-1 | 'Steering Torque', 'LateralDistance', 'psi', 'ay', 'pitch', 'roll', 'HeadHeading', 'Head-Pitch' |
| Driver 6 | 0,7876 | 1-4-2 | 'LateralDistance', 'psi', 'Steering Torque', 'dLongSameAheadNext', 'vyaw', 'Steering-Wheel' |

Half of the test persons are using the standard configuration for the state configuration. Although it is not the preferred configuration in the state search, this indicates that small difference to the ideal state configuration do not influence the result too much, given a good feature selection.

'Lateral Distance' and 'psi' are part of every feature set. The head movement and position is just selected in 50 % of the cases. The number of features is between six and nine features. No correlation between the number of states and the number of features can be detected.

Figure 6.9 shows the development from the generalized model to a standard model which is individually learnt to the personal changes also in the model configuration.

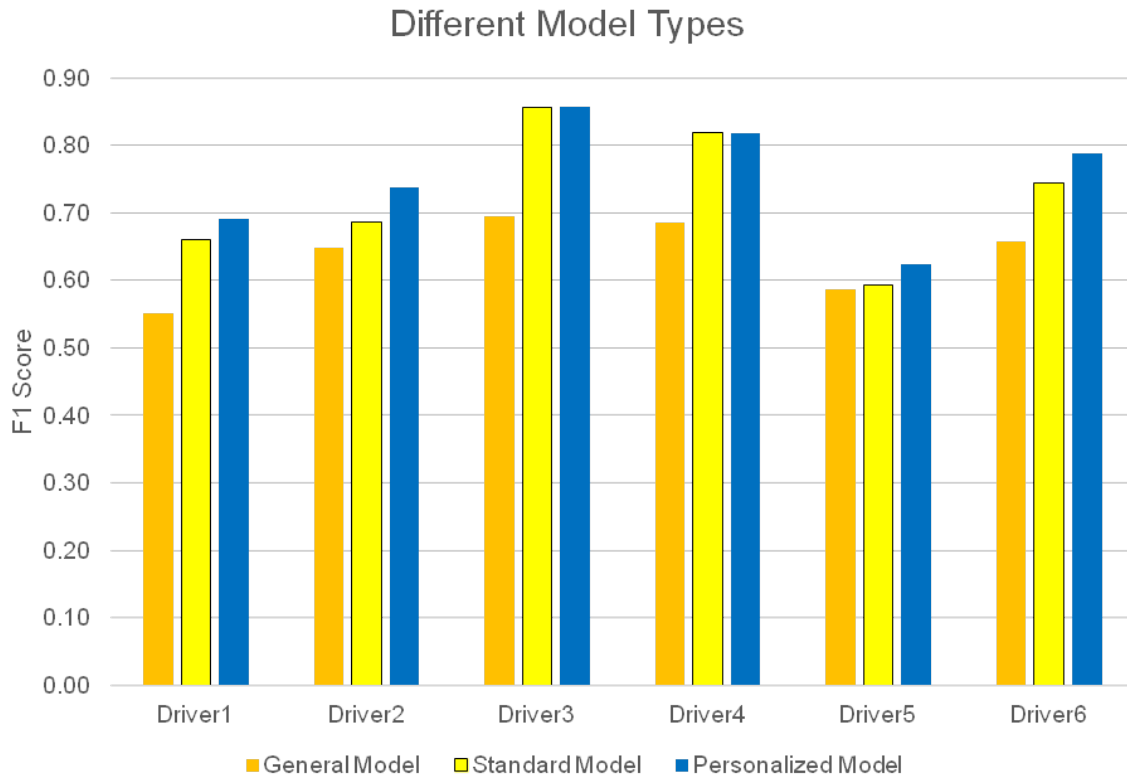


Figure 6.9.: Performance of different models during off-line learning

As mentioned before the data basis for the general model is too small for a fair comparison. So the study of Faller [16] is used to set this in a relation.

Table 6.5.: Off-line learning compared with the literature

| | My Data | Faller's work |
|----------------|--|---|
| General Model | 0,6374 F1 Score 0,056 standard deviation | 0,7132 F1 Score 0,149 standard deviation |
| Personal Model | 0,7525 F1 Score 0,0859 standard deviation | 0,6518 F1 Score 0,136 standard deviation |

So to personalize learning improves the performance of model. Also the standard deviation is lower. Personalized learning improves the model at most if the driver's behaviour is very different from the standard driver. With just six test persons it is unlikely to have someone with extreme different behaviour patterns.

6.2 Incremental Learning

After showing that personalization is a useful step to increase the model performance. The next step is to do this in an on-line scenario, because the individual training can just start after the customer has bought the system. In the next sections, it is shown how the incremental batch learning and the on-line learning after Baldi performs in the lane changing scenario.

6.2.1 Incremental Batch

In the first step the trainings process is analysed. Therefore the data is split into smaller blocks and then the model is updated after every new data block. After every block the performance is evaluated on the data block D4. In the optimal case the model should converge to the same model as the off-line trained on.

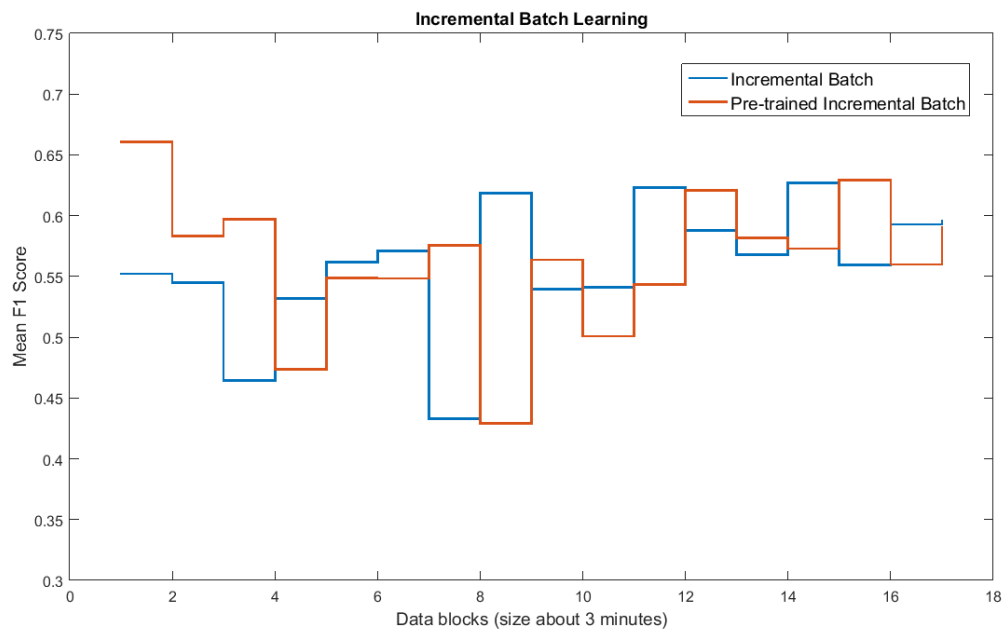


Figure 6.10.: Incremental batch learning, figure shows the mean value over all drivers

For every driver a model is initialized randomly and then the incremental batch learning starts. As second comparison also a pre-trained model is used. For the model configuration the standard model is chosen with a state configuration (1-7-1).

In Figure 6.10 the average over all drivers is shown. A pre-trained model performs better at the beginning but over both types of model converge to a similar F1 score. So in most cases the starting with a pre-trained better would be the better choice. Nevertheless this choice depends on how different the drivers' behaviour is compared with other drivers. For example the start score for driver with id 6 is better with the randomized model.

6.2.2 Baldi

Like in the previous part also the other introduced incremental learning approach is evaluated. As it can be seen in Figure 6.11 the algorithm needs a good initialization to reach a good performance level. The convergence is slower than the incremental batch approach. Also the learning rate is a critical factor. In order to not destroy the good pre-trained model rather small steps are made. This also leads to the problem that with a random start the model is stuck in a local minimum.

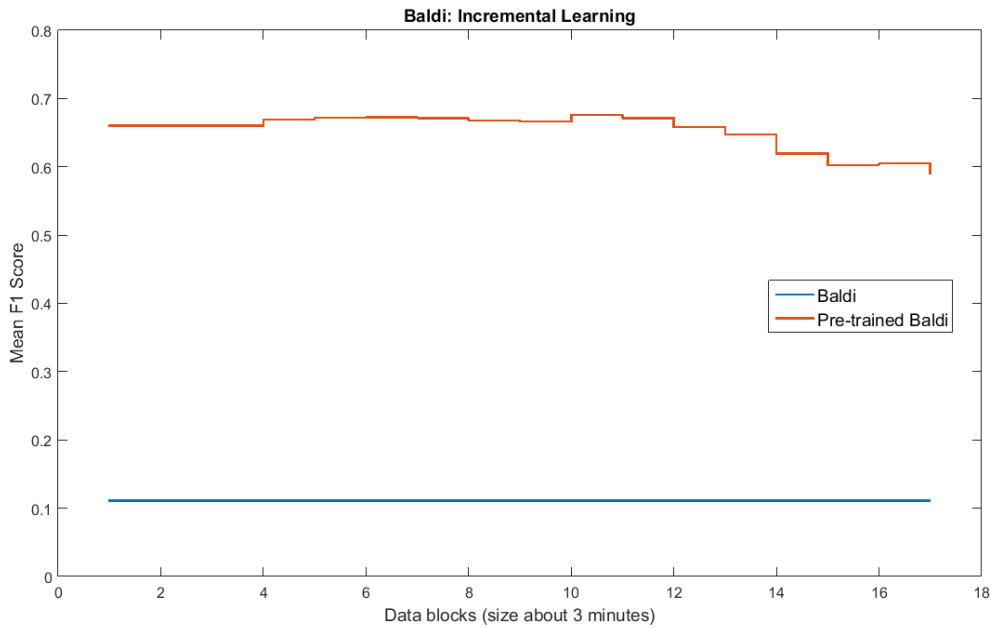


Figure 6.11.: Incremental learning after Baldi: The version with random initialization stays at a low local optimum. The learning rate is too small to escape the local optimum.

A change in the block size of the incoming data shows that the performance increases with larger data blocks. Even the gradient estimate is getting better there are less update rounds. The model with the randomized parameters at the beginning shows not a suitable performance, so for the algorithm after Baldi a good initialization is necessary. The algorithm is adapting to changes slower than the incremental batch version. The decline at the end of the graph is connected with the slower course sections at the end of the complete test drive. When using an optimized learning rate a F1 performance up to 0,30 could be reached but with rather strong fluctuation.

6.3 Model Management

After implementing the model management form chapter 2.4.2, further test can be realised. First the influence of the parameters of the model management to the performance is depicted and in a second part the different strategies from the prediction and selection part are examined. Finally, the impact of the environment to the model performance and incremental learning is shown.

6.3.1 Model Management Parameter

In this part the prediction method and the selection method are fixed. Short term selection and a weighted majority vote are used to make the results comparable.

Update Methods

The four different update methods, incremental batch, baldi, and for each methods also the pre-trained variant, are tested in different combinations. The score is again the F1 score and the evaluation is made on the incoming data blocks.

Table 6.6.: Different update methods

| | incremental Batch | Baldi | pre-trained Batch | pre-trained Baldi | F1Score |
|-------|-------------------|-------|-------------------|-------------------|---------|
| Test1 | x | | | | 0,7318 |
| Test2 | | x | | | 0,1544 |
| Test3 | | | x | | 0,7358 |
| Test4 | | | | x | 0,7125 |
| Test5 | x | x | | | 0,7210 |
| Test6 | | | x | x | 0,7439 |
| Test7 | x | x | x | x | 0,7291 |

Additional a base line model is included. This is a model whose configuration is chosen individual but the model is trained on general data and has never seen the current driver before. To make sure that the model is not just over fitting to the small evaluation data set, also the average model score on the test set D4 is reported.

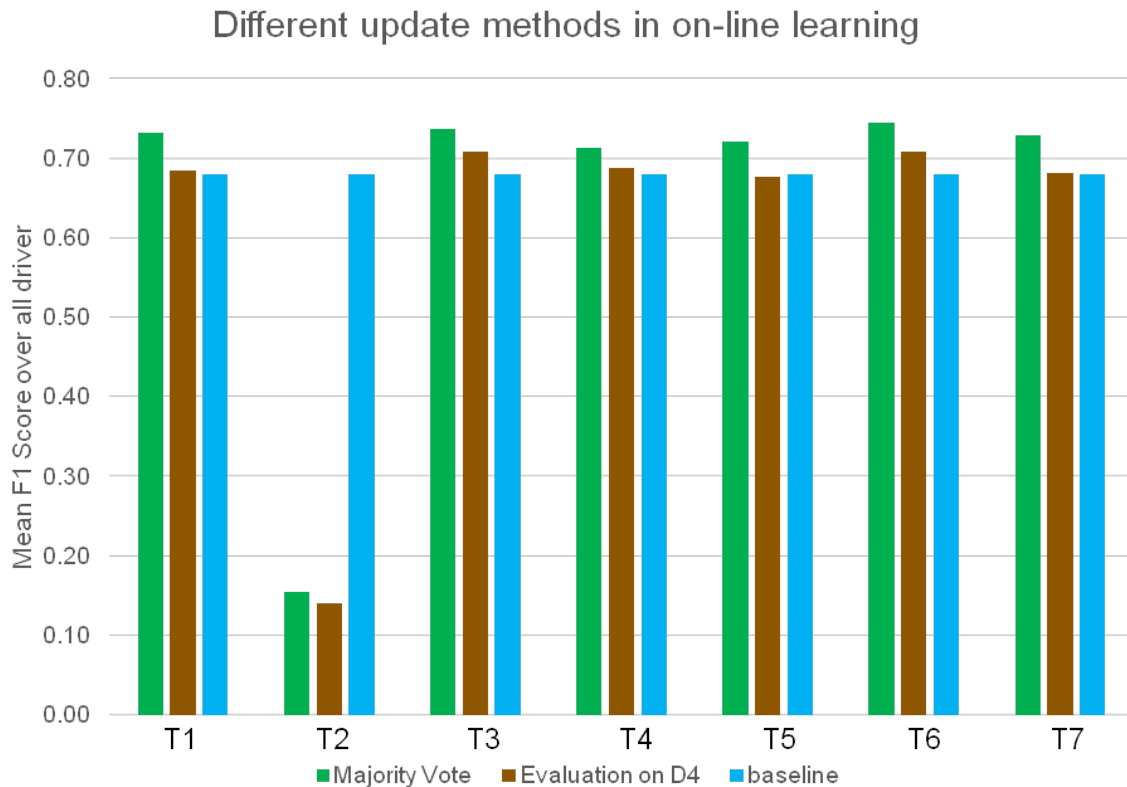


Figure 6.12.: For each update method (see Table 6.6) also different selection strategies are tested. The baseline model is trained off-line.

Also Figure 6.12 shows that the best performance is reached with pre-trained models and using both update methods (T6). Only the Baldi algorithm with no pre-trained initialization performs worse than the base line model. The incremental batch algorithm shows better results than the Baldi algorithm, in both cases random start and pre-trained model. But the combination of pre-trained Baldi and pre-trained batch (T6) outperforms the pre-trained batch only test run (T1). Additionally using the randomized models does not improve the score (T7). The live evaluation on the incoming data shows better scores than the testing on the validation data set D4. But the results between these two evaluation strategies seem to correlate.

Maximum Instances

The number of maximum instances per model class regulates how fast weak models are vanished and with a higher number also the variety within this model class is higher. Here the performance is shown using just two places within each class up to seven free spaces. With more model instances also the computational time increases linear.

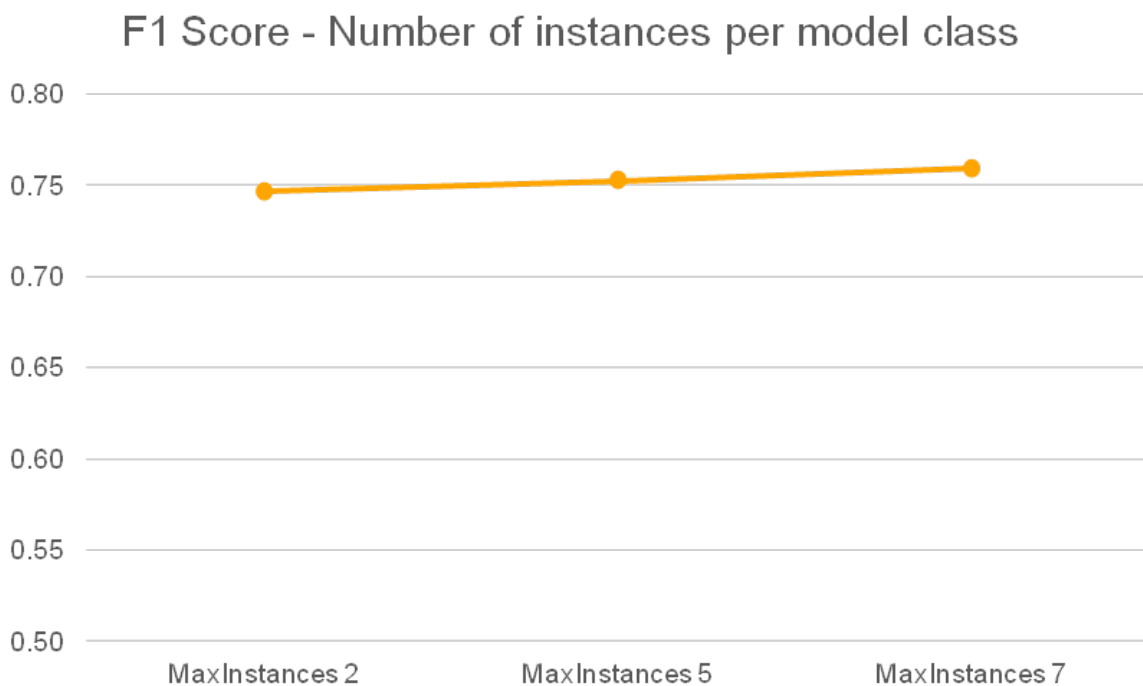


Figure 6.13.: Evaluation score with different limits of maximum instances per model class

As shown in Figure 6.13 on this rather short training data set the number of maximal instances within a class does not have large influence. With more free space the performance is slightly increasing. But compared with the more computational time this gain is rather to be disregarded.

Model Class - Time to live

If a model class is not selected in the selection process step the time to live counter is reduced. Starting with a high counter let weaker model class longer in the model pool. The only advan-

tage of deleting these classes is computational time savings. So there is not influence on the performance that can be measured. Also the period for testing is rather small for such a long term effect.

Model Class Selection

The different model classes allow a kind of state and feature selection during run time. If the state and feature configuration is a personal attribute to a driver over time the model management would select one kind of model configuration. Figure 6.14 shows the selected model classes for prediction. The criteria are an above average performance. The pool of models that is used for prediction differs between the drivers. All model classes are selected at least once, but some model classes are suitable for almost all drivers while other just fit well to one or two drivers. Driver 2 uses all model classes at least once but over time it picks certain classes more often.

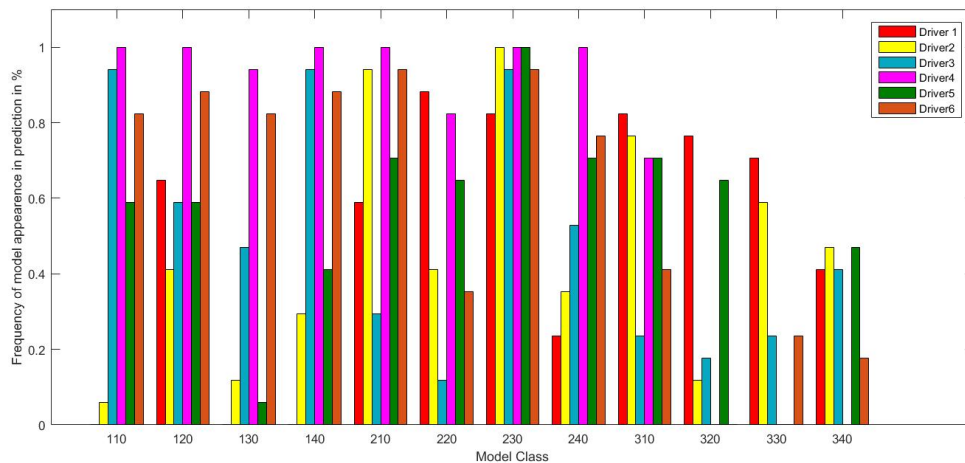


Figure 6.14.: Figure shows which model class is selected how often, the different colors represent the drivers.

So not all drivers need the same model classes but they are also not focused on one or two classes, in average about seven to eight out of twelve classes are selected. The most popular model class 230 using the feature set from the standard model and seven states to model the lane changes. Not so often selected are model classes 320 - 340, these classes using the extended feature set.

6.3.2 Model Selection and Prediction

In this section the selection methods and the final prediction itself are evaluated. A single vote just takes the best model from the model pool and then this model is used for evaluation. However the majority vote takes a bunch of models and then for prediction every model can vote. Close to the different prediction approaches is the question of how to choose the model for the next prediction cycle based on the previous data.

Single Vote

For the single vote scenario two different model selection methods are possible. First the model is chosen by the performance score on the long term evaluation set D4 and in the second case its F1 score on the last short evaluation set is used. The behaviour of the single vote prediction is shown in Figure 6.15. The average score over all drivers is depicted.

The performance of the selection evaluated on the long term data is better, here the curve is smoother and has less out-liners.

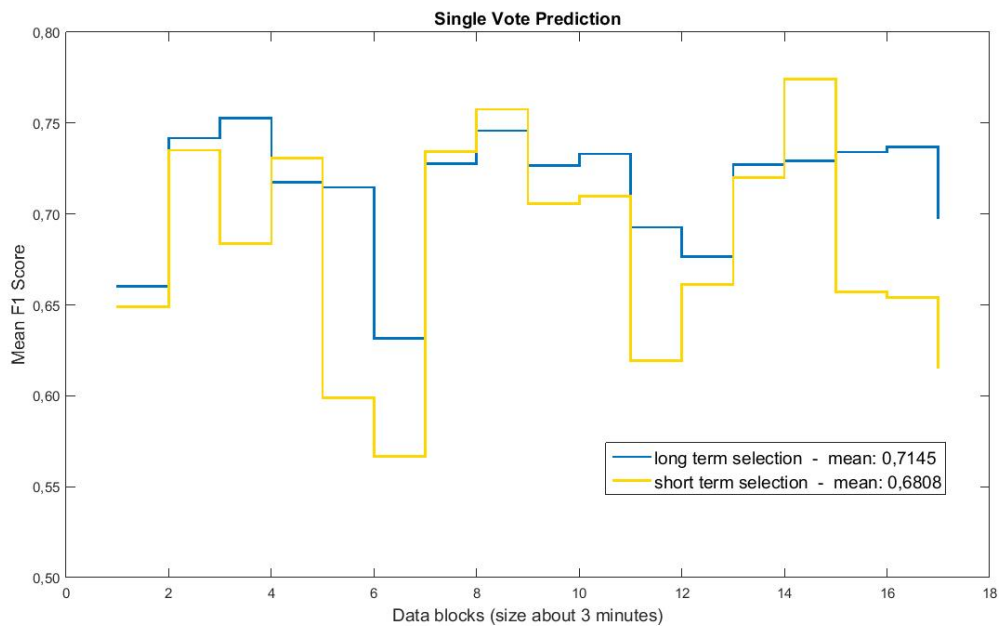


Figure 6.15.: Single vote with different selection methods

The update methods do not infect the single voting significant (see Appendix D.1. Only exception is the score from Baldi's method without pre-training, but this low score is comparable with the majority vote score for this method.

Majority Vote

There are two versions of majority votes, on the one hand unweighed majority vote and on the other hand weighted majority vote. Because of the pre-selection of models the difference between these two is rather small. Weighted majority vote outperforms the un-weighted version in all parts of the course. The decreases in certain parts are deeper when all models have equal weights.

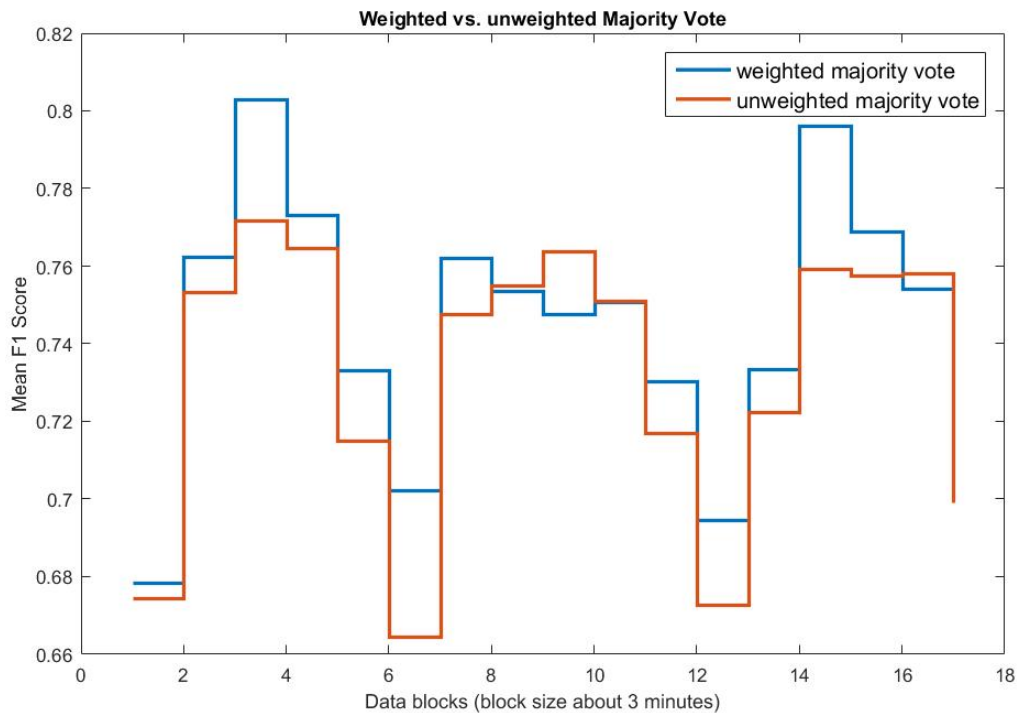


Figure 6.16.: Majority vote with and without weights

Given that the evaluation data blocks are similar to the previous data blocks the weighted majority vote shows the best performance and should be used.

Long and short term selection

Using the short term evaluation as selection method has the advantage that the model can react to temporary changes in driving or the environment. But corrupt data also decrease the model performance for the next data block. To be able to react to both scenarios also a mixture of both is implemented. The performance of these approaches can be seen in Fig 6.17.

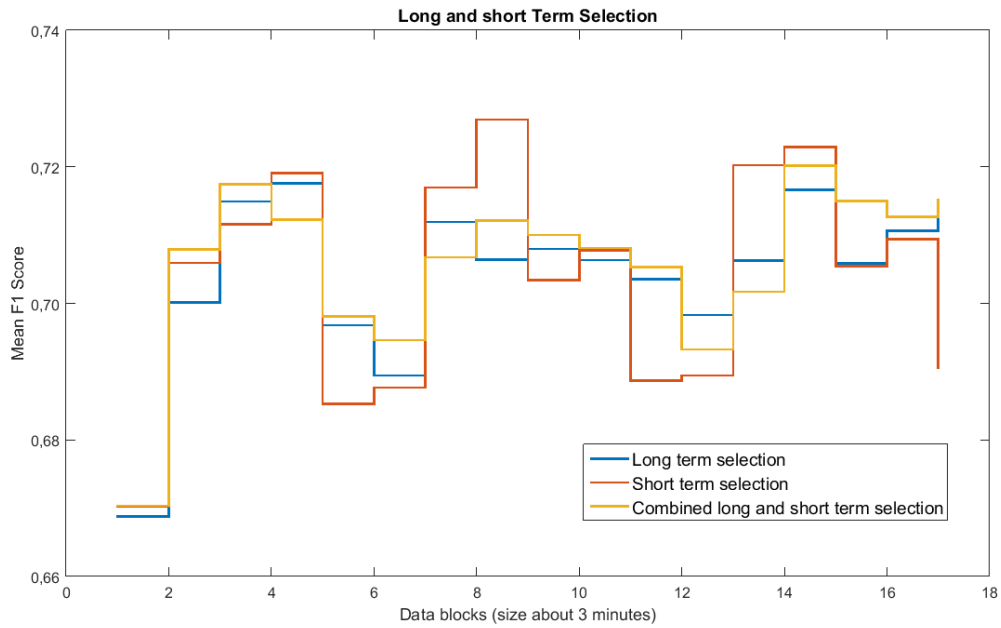


Figure 6.17.: Different selection strategies before the final prediction. The evaluation could be either on the validation set D4 or on the current incoming data block. In yellow a combination of the long and short term evaluation for selection is shown.

The short term selection leads to higher peaks in the graph while the combination of long and short term selection produce the most stable results. The total score over all drivers is quite similar to all approaches in this test setting. A combination of long and short term evaluation methods reaches the best score. Here the model predicts with a F1 score of 76,49 %.

To summarize the findings so far, Figure 6.18 shows the performance of single vote with long term selection and the majority vote with long and short term selection. Furthermore, a baseline model is included which has no on-line learning activities.

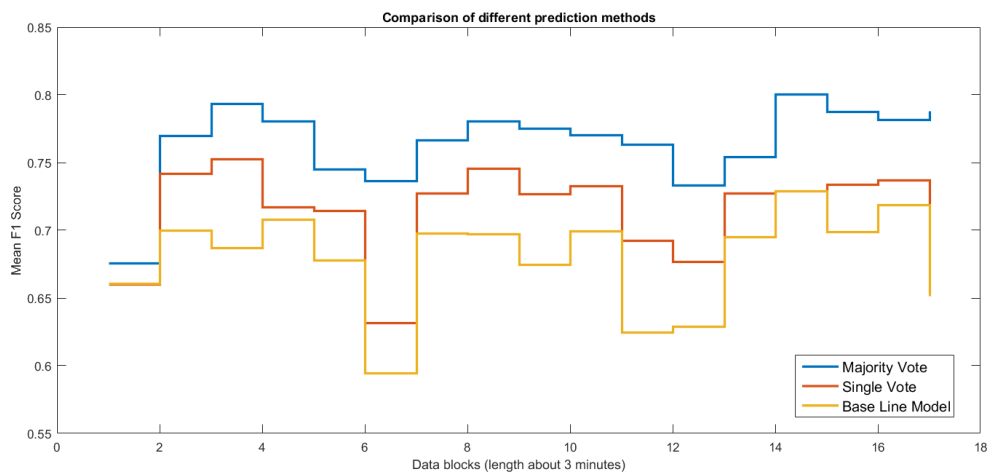


Figure 6.18.: Summary of different prediction techniques

The best results are reached with the weighted majority vote. As a selection method the combination of long and short term selection is preferred. In all cases the basis model without on-line learning is the worst model.

6.3.3 Environment influence and Data

Further aspects of the performance of the prediction model are examined.

Data Blocks

The size of the data blocks is varied from about 3 minutes up to 20 minutes for each block. No big difference could be detected in the mean of the F1 score over all drivers. The tests are made with weighted majority vote and short term evaluation in the selection process.

Table 6.7.: Different block length

| block size | 3 min | 5 min | 10 min | 20 min |
|------------|--------|--------|--------|--------|
| F1 score | 0,7439 | 0,7412 | 0,7500 | 0,7440 |

Resort Data

The different section on the highway may cause changes in the model parameters. If the data is sort in a way that similar sections are close together the total performance increases.

The hypothesis that the model is can adapt faster to a new highway section could not be clearly seen in the data. It was expected that with a new highway section first the model performance is falling down and then slow increasing again. Probably because of the high variance of the performance in the short evaluation data sets this effect is not good to detect. Nevertheless, at the transition of different sections there are jumps in the model performance especially at the beginning. Later on the changes are lower, which could indicate a more generalized model.

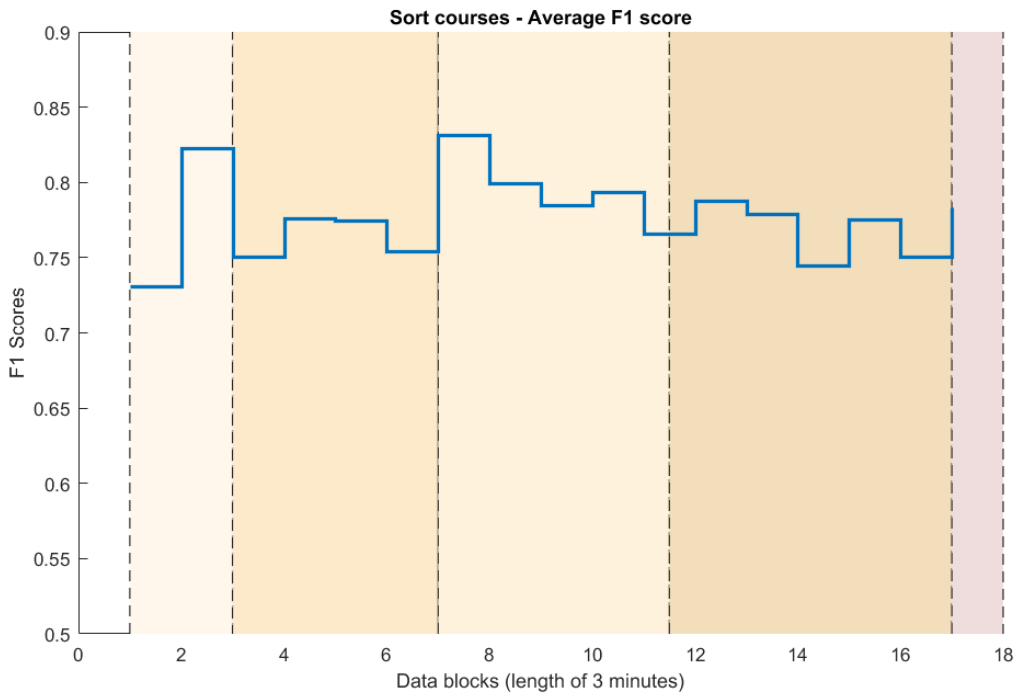


Figure 6.19.: Resort data and the average F1 performance. The background colour indicates the different course parts

Nevertheless the average performance is better with the sorted parts, so fewer jumps in the highway profile are better. But just certain drivers can profit from the resorted data set as seen in Figure 6.20.

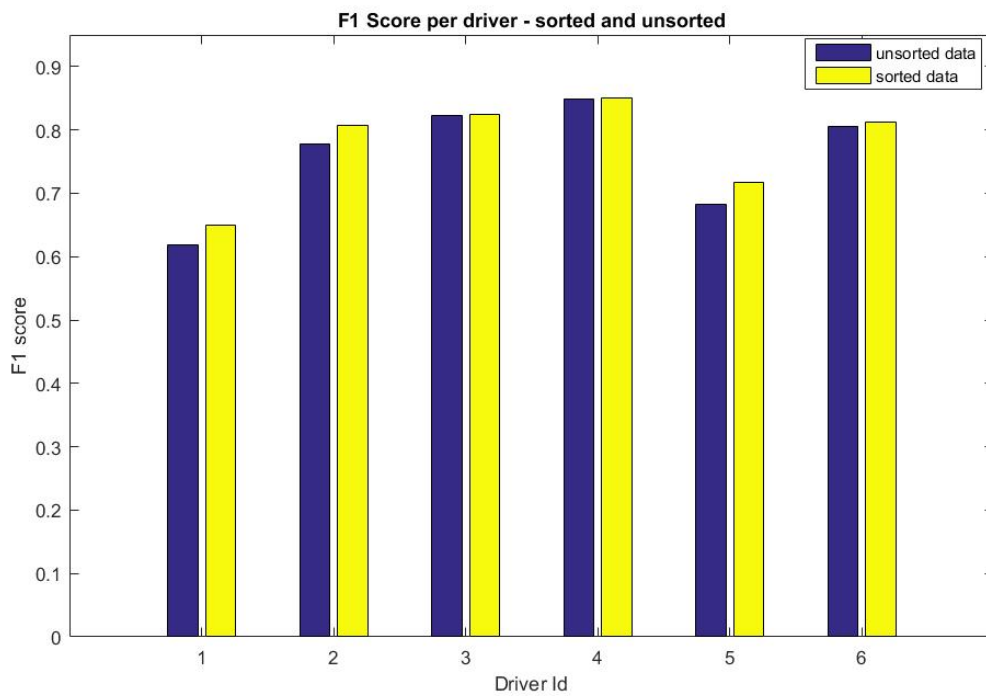


Figure 6.20.: Driver on sorted and unsorted course parts

7 Conclusion and Future Work

In this work a new data set for predicting the lane change intention has been recorded. This data set is used to find personalized model while all data is available. Compared to the general model a training process that just considers the individual data improves the model score the most. Further adaptation on the state configuration just slightly improves the performance. The set of relevant features differs from driver to driver. The vehicle dynamics are the dominate feature group for most drivers, especially the distance to the middle lane. From the camera system the head movement is the most important feature. Concerning the feature selection methods SFFS founds the best results. Personalized models also perform better than comparable tests in literature. Problems why the score is not even better may be caused by the model type itself and the labelling method.

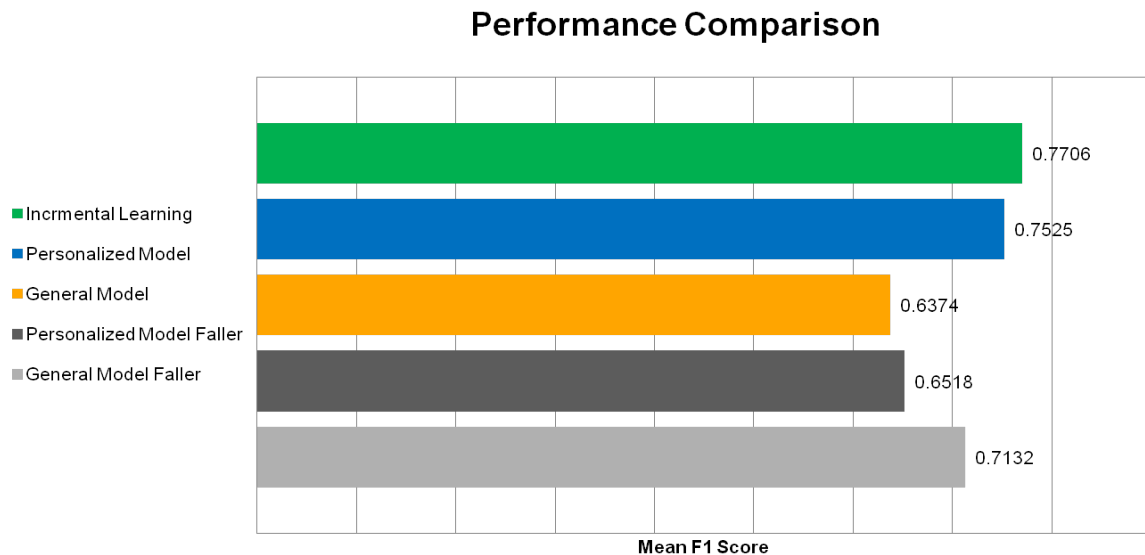


Figure 7.1.: Comparison of the different approaches

With an incremental learning approach the same and even better results could be achieved. This method has two advantages first less data is needed to reach this point and second the training process could be done during driving. Small blocks of about 3 minutes with 4-6 lane changes are needed.

In order to reach this goal a general model management framework is implemented. The classification task now is solved with some Ensemble Learning methods. The best results are gained with a weighted majority vote. For the on-line update methods are incremental batch learning works fine. The data blocks are large enough and the process is more stable than updating the model after every single data point.

An ensemble of models allows implementing different strategies to ensure a constant model quality even with changing data. A good trade of between robustness and fast adaptation to changes is given with a pre-selection strategy that evaluates the model on the short term data and on a fixed long term data set.

An important aspect is that the model performance differs on traffic situation. Slower or faster highway section and less or more traffic on the road have a significant influence on the performance.

7.1 Future Work

Further ideas could be implemented to try to get an even better model performance. The adaptation to the traffic situation should be examined in more detail, therefore a larger data set is needed. It is conceivable to have a filter before the actual prediction according to the current traffic situation, for example current speed of the car. After passing this filter the right model type is selected. Also a feature and state selection during run time could be considered. More data is also needed to test the performance of the current approach concerning different types of drivers, here more drivers would be better in order to have more driving behaviour apart the average. Longer test rides are necessary to find out the limits of all parameters of the model management. But the general model management also could be tried out with a different kind of basis model for example neural networks. To bring the system from the simulator to a real car environment the process of collecting data has to be adapted. Probably the data is recorded until a certain number of lane changes have been performed.

Not regarded in this work is the idea of solving the on-line learning task through cloud computing. Sending all the data to the cloud would solve the storage and computational problems.

Appendices

A Feature Selection

| | Feature |
|----|----------------------|
| 1 | AcceleratorPedal |
| 2 | Steering Angle |
| 3 | Steering Torque |
| 4 | LateralDistance |
| 5 | psi |
| 6 | ax |
| 7 | ay |
| 8 | pitch |
| 9 | roll |
| 10 | v |
| 11 | vyaw |
| 12 | yaw |
| 13 | dLongSameAheadNext |
| 14 | dLongSameBehindNext |
| 15 | dLatSameBehindNext |
| 16 | HeadHeading |
| 17 | HeadPitch |
| 18 | HeadRoll |
| 19 | GazePitch |
| 20 | EstimatedGazeHeading |
| 21 | HeadPositionX |
| 22 | HeadPositionY |
| 23 | HeadPositionZ |

Figure A.1.: Pre-selected Features

B Incremental Hidden Markov Models

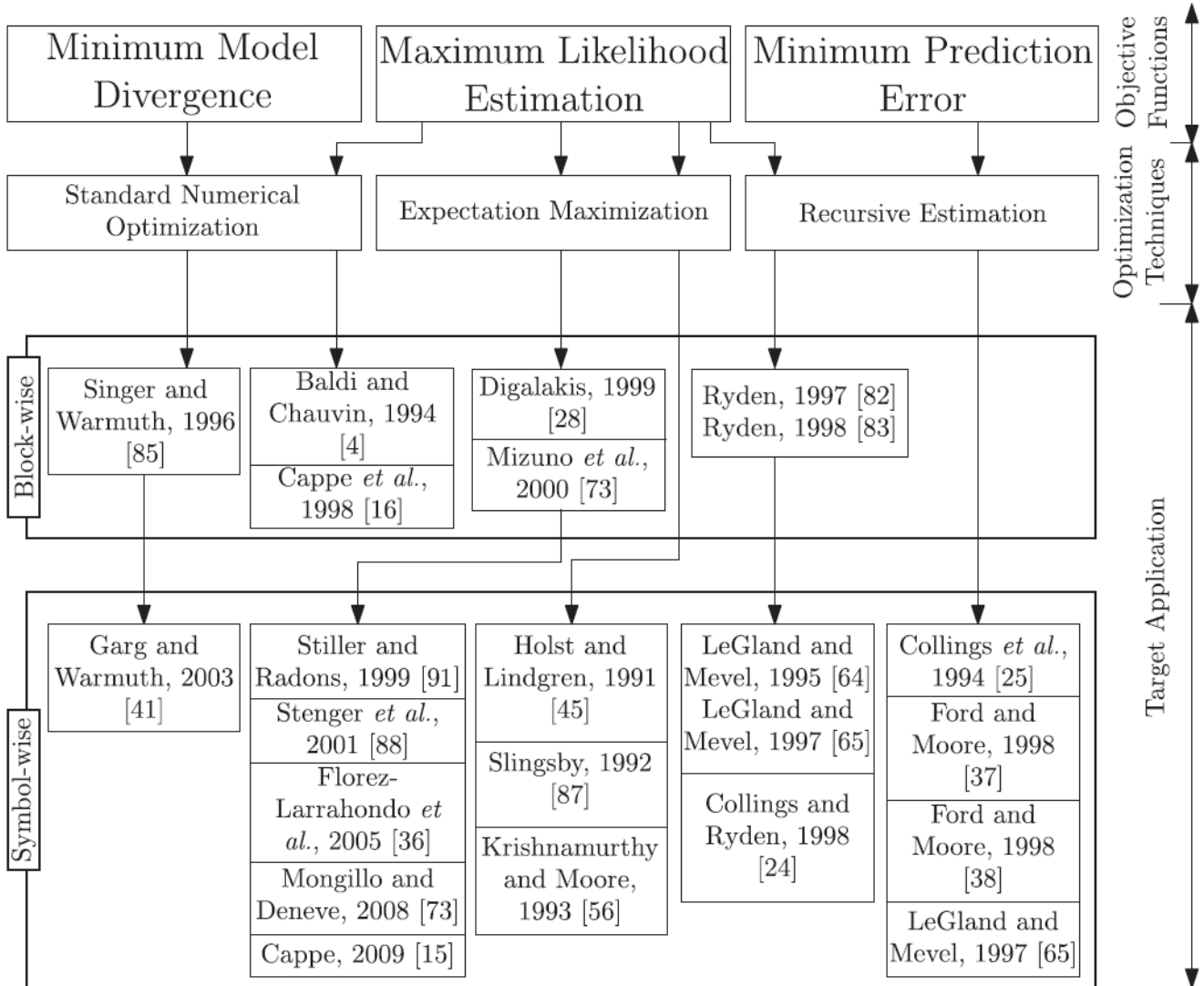










Figure B.1.: Incremental Hidden Markov Models after [23]

C Evaluation

| Bezeichnung | Vorhersage | Label |
|------------------------------------|---|---|
| › True Predictions (tp) |  |  |
| › False Predictions (fp) |  |  |
| › False Positive Predictions (fpp) |  |  |
| › Missed Predictions (mp) |  |  |

| Precision | Recall | F1 |
|-----------------------------|----------------------------|--|
| $Pr = \frac{tp}{tp+fp+fpp}$ | $Re = \frac{tp}{tp+fp+mp}$ | $F1 = \frac{2 \cdot Pr \cdot Re}{Pr + Re}$ |

Figure C.1.: Calculating the F1 score after [16]

D Single Voting

Table D.1.: Different update methods with Single Voting

| | iBatch | Baldi | preBatch | preBaldi | F1Score(singleVote) |
|-------|--------|-------|----------|----------|---------------------|
| Test1 | x | | | | 0,6881 |
| Test2 | | x | | | 0,1811 |
| Test3 | | | x | | 0,6709 |
| Test4 | | | | x | 0,6934 |
| Test5 | x | x | | | 0,6867 |
| Test6 | | | x | x | 0,6478 |
| Test7 | x | x | x | x | 0,6844 |

List of Figures

| | |
|--|----|
| 2.1. Machine Learning Process [1] | 7 |
| 2.2. Confusion Matrix for a classification task | 10 |
| 2.3. Example of the structure of a Hidden Markov Model with three hidden states and a Gaussian emission probability | 12 |
| 2.4. Standard Batch Learning: The data pool is increasing over time. At every time step the model is learnt from the beginning | 16 |
| 2.5. On-line Learning: The model parameters are updated with every new data point, just the model parameters are saved. | 17 |
| 2.6. Incremental Learning: The learning process can iterate over a block of data several times like in batch learning. The model parameters are saved and used as a start point in the next iteration. | 17 |
| 2.7. On-line learning for HMM according to Cappé [7] | 19 |
| 2.8. Components of a Model Management after [24] | 20 |
| 3.1. Lane Change Classification with a mulit-class Hidden Markov Model | 23 |
| 4.1. Development F1 Score compared to the number of lane changes in the training data | 25 |
| 4.2. Figure shows the steps of the lane change manoeuvre and how to enforce more overtake manoeuvres | 26 |
| 4.3. Different sections of the test course. Detailed description for each section could be found in the table above. | 27 |
| 5.1. Concept SFFS: First try to add the best fitting feature to the selected features, then test if a removal of one feature could increase the score. | 31 |
| 5.2. Pseudo Code for SFBS implementation | 32 |
| 5.3. Sliding Window: Development of the data blocks over time. | 34 |
| 5.4. Different evaluation strategies | 34 |
| 5.5. On-line learning according to Cappé in a simple example: Mean values for the Gaussian emission probabilities are shown. From a known HMM the data is sampled and then used in training process. | 35 |
| 5.6. Parameter Search: This figure should illustrate that for a driver with a parameter set far away from the average, it may be better to randomly start the parameter search instead of using the general model as a starting point. | 36 |
| 5.7. Pseudo Code Model Management | 38 |
| 5.8. Combination of model classes, each class is defined by its feature and state configuration | 39 |
| 5.9. Data management for the Model Management, new data is first used for testing the models and then for training in the next time step. | 41 |
| 5.10. Update process for the model management: model score and model parameters are updated. | 42 |

| | |
|--|----|
| 6.1. Data is split into four parts, each block of data is about 20 minutes long. | 45 |
| 6.2. For each driver a general model is learnt on the data of the five other drivers and then evaluated. | 46 |
| 6.3. Personalized Models compared to the general Model | 47 |
| 6.4. Matrix of selected features through SFBS - in grey the features from the standard model are marked | 49 |
| 6.5. Performance development after SFFS feature selection compared to the standard model | 50 |
| 6.6. Matrix of selected features through SFFS. In grey the features from the standard model are marked. | 51 |
| 6.7. Ratio between deviation to average F1 score and number of states - higher values indicate a good performance with minimum number of states. | 53 |
| 6.8. Combine the results from feature and state selection | 54 |
| 6.9. Performance of different models during off-line learning | 56 |
| 6.10. Incremental batch learning, figure shows the mean value over all drivers | 57 |
| 6.11. Incremental learning after Baldi: The version with random initialization stays at a low local optimum. The learning rate is too small to escape the local optimum. | 58 |
| 6.12. For each update method (see Table 6.6) also different selection strategies are tested. The baseline model is trained off-line. | 59 |
| 6.13. Evaluation score with different limits of maximum instances per model class | 60 |
| 6.14. Figure shows which model class is selected how often, the different colors represent the drivers. | 61 |
| 6.15. Single vote with different selection methods | 62 |
| 6.16. Majority vote with and without weights | 63 |
| 6.17. Different selection strategies before the final prediction. The evaluation could be either on the validation set D4 or on the current incoming data block. In yellow a combination of the long and short term evaluation for selection is shown. | 64 |
| 6.18. Summary of different prediction techniques | 64 |
| 6.19. Resort data and the average F1 performance. The background colour indicates the different course parts | 66 |
| 6.20. Driver on sorted and unsorted course parts | 66 |
| 7.1. Comparison of the different approaches | 67 |
| A.1. Pre-selected Features | 70 |
| B.1. Incremental Hidden Markov Models after [23] | 71 |
| C.1. Calculating the F1 score after [16] | 72 |

List of Tables

| | |
|--|----|
| 4.1. Different test run sections | 27 |
| 5.1. Search space state selection | 30 |
| 5.2. Search space state selection | 33 |
| 6.1. Most frequently chosen Features | 51 |
| 6.2. State Selection | 52 |
| 6.3. State Selection with reduced complexity | 53 |
| 6.4. Best Performance - Personalized Off-line Training | 55 |
| 6.5. Off-line learning compared with the literature | 56 |
| 6.6. Different update methods | 59 |
| 6.7. Different block length | 65 |
| D.1. Different update methods with Single Voting | 73 |

Bibliography

- [1] Muhammed Ayfan. Machine learning. <http://www.ineffable.in/technology/database/machine-learning/>, 2016.
- [2] Taiwo Oladipupo Ayodele. *Types of machine learning algorithms*. INTECH Open Access Publisher, 2010.
- [3] Pierre Baldi and Yves Chauvin. Smooth on-line learning algorithms for hidden markov models. *Neural Computation*, 6(2):307–318, 1994.
- [4] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.
- [5] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.
- [6] Johnell O Brooks, Richard R Goodenough, Matthew C Crisler, Nathan D Klein, Rebecca L Alley, Beatrice L Koon, William C Logan, Jennifer H Ogle, Richard A Tyrrell, and Rebekkah F Wills. Simulator sickness during driving simulation studies. *Accident Analysis & Prevention*, 42(3):788–796, 2010.
- [7] Olivier Cappé. Online em algorithm for hidden markov models. *Journal of Computational and Graphical Statistics*, 2012.
- [8] Paulo R Cavalin, Robert Sabourin, Ching Y Suen, and Alceu S Britto Jr. Evaluation of incremental learning algorithms for hmm in the recognition of alphanumeric characters. *Pattern Recognition*, 42(12):3241–3253, 2009.
- [9] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.
- [10] Hal Daumé III. A course in machine learning. *chapter*, 5:69, 2012.
- [11] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.
- [12] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.
- [13] Vasilios V Digalakis. Online adaptation of hidden markov models using incremental estimation algorithms. *IEEE Transactions on Speech and Audio Processing*, 7(3):253–261, 1999.
- [14] Chenxi Ding, Wuhong Wang, Xiao Wang, and Martin Baumann. A neural network model for driver’s lane-changing trajectory prediction in urban traffic flow. *Mathematical Problems in Engineering*, 2013, 2013.

-
- [15] Anup Doshi and Mohan Trivedi. A comparative exploration of eye gaze and head motion cues for lane change intent prediction. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 49–54. IEEE, 2008.
- [16] Fabian Faller. Utilization of a hidden-markov-model for the prediction of lane change maneuvers. Master’s thesis, TU Darmstadt, Knowledge Engineering Group, March 2016. Master’s Thesis.
- [17] German Florez-Larrahondo, Susan Bridges, and Eric A Hansen. Incremental estimation of discrete hidden markov models based on a new backward procedure. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 758. Menlo Park, CA; Cambridge, MA; London; AAI Press; MIT Press; 1999, 2005.
- [18] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.
- [19] Nicholas J Higham. Computing the nearest correlation matrix—a problem from finance. *IMA journal of Numerical Analysis*, 22(3):329–343, 2002.
- [20] Ashesh Jain, Hema S Koppula, Bharad Raghavan, Shane Soh, and Ashutosh Saxena. Car that knows before you do: Anticipating maneuvers via learning temporal driving models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3182–3190, 2015.
- [21] Ashesh Jain, Avi Singh, Hema S Koppula, Shane Soh, and Ashutosh Saxena. Recurrent neural networks for driver activity anticipation via sensory-fusion architecture. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3118–3125. IEEE, 2016.
- [22] Wael Khreich, Eric Granger, Ali Miri, and Robert Sabourin. A comparison of techniques for on-line incremental learning of hmm parameters in anomaly detection. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–8. IEEE, 2009.
- [23] Wael Khreich, Eric Granger, Ali Miri, and Robert Sabourin. A survey of techniques for incremental learning of hmm parameters. *Information Sciences*, 197:105–130, 2012.
- [24] Wael Khreich, Eric Granger, Robert Sabourin, and Ali Miri. Combining hidden markov models for improved anomaly detection. In *2009 IEEE International Conference on Communications*, pages 1–6. IEEE, 2009.
- [25] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.
- [26] Vikram Krishnamurthy and John B Moore. On-line estimation of hidden markov model parameters based on the kullback-leibler information measure. *IEEE Transactions on signal processing*, 41(8):2557–2573, 1993.
- [27] Mineichi Kudo and Jack Sklansky. Comparison of algorithms that select features for pattern classifiers. *Pattern recognition*, 33(1):25–41, 2000.

-
- [28] Puneet Kumar, Mathias Perrollaz, Stéphanie Lefevre, and Christian Laugier. Learning-based approach for online lane change intention prediction. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 797–802. IEEE, 2013.
- [29] Stéphanie Lefevre, Yiqi Gao, Dizan Vasquez, H Eric Tseng, Ruzena Bajcsy, and Francesco Borrelli. Lane keeping assistance with learning-based driver model and model predictive control. In *12th International Symposium on Advanced Vehicle Control*, 2014.
- [30] Kevin Markham. Simple guide to confusion matrix terminology. <http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>, 2016.
- [31] Joel C McCall and Mohan M Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE transactions on intelligent transportation systems*, 7(1):20–37, 2006.
- [32] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
- [33] Gianluigi Mongillo and Sophie Deneve. Online learning with hidden markov models. *Neural computation*, 20(7):1706–1716, 2008.
- [34] Brendan Morris, Anup Doshi, and Mohan Trivedi. Lane change intent prediction for driver assistance: On-road design and evaluation. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 895–901. IEEE, 2011.
- [35] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [36] Shai Shalev-Shwartz and Yoram Singer. *Online learning: Theory, algorithms, and applications*. 2007.
- [37] Yoram Singer and Manfred K Warmuth. Training algorithms for hidden markov models using entropy based distance functions. In *NIPS*, pages 641–647. Citeseer, 1996.
- [38] Alex Smola and SVN Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32:34, 2008.
- [39] Ranjeet Singh Tomar, Shekhar Verma, and Geetam Singh Tomar. Prediction of lane change trajectories through neural network. In *Computational Intelligence and Communication Networks (CICN), 2010 International Conference on*, pages 249–253. IEEE, 2010.
- [40] Dizan Vasquez, Thierry Fraichard, and Christian Laugier. Incremental learning of statistical motion patterns with growing hidden markov models. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):403–416, 2009.
- [41] Hanhong Xue, Venu Govindaraju, et al. Hidden markov models combining discrete symbols and continuous attributes in handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(3):458–462, 2006.