From logistic regression to self-driving cars: Chances and challenges of using machine learning for highly automated driving
From logistic regression to self-driving cars

Machine learning has been one of the hottest topics in research and industry in the last couple of years. Renewed attention has resulted from the latest advancements in computational performance and algorithms compared with the advent of machine learning decades ago. Recent impressive results in artificial intelligence have been facilitated by machine learning, particularly by deep learning solutions. Applications include natural language processing (NLP), personal assistance, the victory of AlphaGo over a human being, and the achievement of human level behavior in learning to play Atari games.

Considering that machine learning and deep learning enable such impressive results in tackling extremely complex problems, it is obvious that researchers and engineers consider to also apply them in the context of highly automated driving (HAD) scenarios towards self-driving cars. The first promising results have been achieved in this area with NVIDIA’s Davenet, Comma.ai, Google Car, and Tesla. Machine learning and deep learning approaches have resulted in initial prototypes, but the industrialization of such functionalities poses additional challenges with regard to essential functional safety considerations, for example.

This article aims to contribute to the ongoing discussions about the role of machine learning in the automotive industry and to highlight the importance of this topic in the context of self-driving cars. In particular, it aims to increase understanding of the capabilities and limitations of machine learning technologies.

This article first of all discusses the design space and architectural alternatives for machine learning-based highly automated driving considering also the EB robinos reference architecture. Two selected use cases that are current in research and development at Elektrobit are then presented in detail. Section 3 provides the theoretical background to machine learning and deep neural networks (DNN) that is the basis for deriving criteria for selecting a machine learning approach according to the given task. Finally, this article discusses the verification and validation challenges that affect functional safety considerations.

Machine learning and highly automated driving

It is a complex and nontrivial task to develop the highly automated driving functionalities that lead to self-driving cars. Engineers typically tackle such challenges using the principle of divide and conquer. This is for a good reason: A decomposed system with clearly defined interfaces can be tested and verified much more thoroughly than a single black box.

Our approach to highly automated driving is EB robinos, depicted in figure 1, page 3. EB robinos is a functional software architecture with open interfaces and software modules that enables to manage the complexity of autonomous driving. The EB robinos reference architecture integrates the components following the sense, plan, act decomposition paradigm. Moreover, it also makes use of machine learning technology within its software modules in order to cope with the highly unstructured real-world driving environment. The subsections below contain selected examples of the technologies that are investigated within EB robinos.

In contrast, end-to-end deep learning approaches also exist, which span everything from sense to act (Bojarski et al. 2016). However, with respect to handling and training of corner cases and rare events, and with regards to the exponential amount of training data necessary, a decomposition approach (i.e., semantic abstraction) is considered as more reasonable (Shalev-Shwartz et al. 2016).

Nevertheless, a decision about which parts are better tackled in isolation to others or in combination with others is required even if the decomposition approach is followed. It is also necessary to determine whether a machine learning approach is expected to outperform a traditionally engineered algorithm for the task accomplished by a particular block. Not least, this decision may be influenced by functional safety considerations. Functional safety is a crucial element of autonomous driving, as described in section 4 of this article. Traditional software components are written on the basis of concrete requirements and are tested accordingly.
This project proposes a speed limit and end of restriction traffic sign (TS) recognition system in the context of enhancing OpenStreetMap (OSM) data used in entry navigation systems. The aim is to run the algorithm on a standard smartphone that can be mounted on the windshield of a car. The system detects traffic signs along with their GPS position and uploads the collected data to backend servers via the mobile data connection of the phone. The approach is divided mainly into two stages: detection and recognition. Detection is achieved through a boosting classifier. Recognition is performed through a probabilistic Bayesian inference framework that fuses information delivered by a collection of visual probabilistic filters. Section 3 of this article contains a description of the theoretical background behind the used algorithms. Figure 2 depicts the block diagram of the traffic signs recognition (TSR) algorithm.

The color image obtained is passed to the detector in 24-bit RGB format. The detection process is carried out by evaluating the response of a cascade classifier calculated through a detection window.

The main issues in the testing and validation of machine learning systems are their black box nature and the stochastic behavior of the learning methods. It is basically impossible to predict how the system learns its structure.

The criteria and theoretical background given above can provide guidance for informed decisions. EB is currently researching and developing use cases in which machine learning approaches are considered to be promising. Two such use cases are presented next. The first deals with the generation of artificial training samples for machine learning algorithms and their deployment for traffic sign recognition. The second use case describes our approach to self-learning cars. Both examples make use of current cutting-edge deep learning technology.

**Use case 1: Artificial sample generation and traffic sign recognition**

This project proposes a speed limit and end of restriction traffic sign (TS) recognition system in the context of enhancing OpenStreetMap (OSM) data used in entry navigation systems. The aim is to run the algorithm on a standard smartphone that can be mounted on the windshield of a car. The system detects traffic signs along with their GPS position and uploads the collected data to backend servers via the mobile data connection of the phone. The approach is divided mainly into two stages: detection and recognition. Detection is achieved through a boosting classifier. Recognition is performed through a probabilistic Bayesian inference framework that fuses information delivered by a collection of visual probabilistic filters. Section 3 of this article contains a description of the theoretical background behind the used algorithms. Figure 2 depicts the block diagram of the traffic signs recognition (TSR) algorithm.

The color image obtained is passed to the detector in 24-bit RGB format. The detection process is carried out by evaluating the response of a cascade classifier calculated through a detection window.

**Figure 1: Open EB robinos reference architecture**

**Figure 2: Block diagram of the smartphone-based TSR system**
This detection window is shifted across the image at different scales. The probable traffic sign regions of interest (RoI) are collected as a set of object hypotheses. The classification cascade is trained with extended local binary patterns (eLPB) from the point of view of feature extraction. Each element in the hypotheses vector is classified into a traffic sign by a support vector machine (SVM) learner.

Traffic sign recognition methods rely on manually labelled traffic signs, which are used to train both the detection and the recognition classifiers. The labelling process is tedious and prone to error due to the variety of traffic sign templates used in different countries. Figure 3 shows the differences in some speed limit signs.

Specific training data for each country is required for the traffic sign recognition method to perform well. It is time-consuming to create enough manually labelled traffic signs because position, illumination, and weather conditions have to be taken into account. We have created an algorithm that generates training data automatically from a single artificial template image to overcome the challenge of manually annotating large numbers of training samples. Figure 4 shows the structure of the algorithm.

This approach provides a method for generating artificial data that is used in the training stages of machine learning algorithms. The method uses the reduced dataset of real and generic traffic sign image templates for each country to output a collection of images. The features of these images are artificially defined by a sequence of image template deformation algorithms. The artificial images thus obtained are evaluated against a reduced set of real-world images using kernel principal components analysis (KPCA). The artificial data set is suitable for the training of machine learning systems, in this particular case for traffic sign recognition, when the characteristics of the generated images correspond to those of the real images.

We replaced the Boosting SVM classifiers with a deep region-based detection and recognition convolutional neural network to improve the precision of the original traffic sign recognition system. The network is deployed using Caffe (Jia et al. 2014), which is a deep neural network library developed by Berkley and supported by NVIDIA. Caffe is a pure C++/CUDA library with Python and Matlab interfaces. In addition to its core deep learning functionalities, Caffe also provides reference deep learning models that can be used directly in machine learning applications. Figure 5, page 5, shows the Caffe net structure used for traffic sign detection and recognition. The different, colored blocks represent convolution (red), pooling (yellow), activation (green), and fully connected network layers (purple).
Figure 5: Deep region-based detection and recognition convolutional neural network in Caffe
Use case 2: Learning how to drive

The revolution in deep learning has recently increased attention on another paradigm, which is referred to as reinforcement learning (RL). In RL, an agent by itself learns how to perform certain tasks by means of a reward system. The methodology is in the category of semi-supervised learning because the design of the reward system requires domain-specific knowledge. That is even though there is no required labeling for the input data, in contrast with supervised learning. This recent interest in RL is due mainly to the seminal work of the Deep Mind team. This team managed to combine RL with a deep neural network capable of learning the action value function (Mnih et al. 2016). Their system was able to learn to play several Atari games at human level capacity.

We constructed the deep reinforcement learning system, shown in figure 6, page 7, in order to experiment safely with autonomous driving learning. This system uses the TORCS open-source race simulator (Wymann et al. 2014). TORCS is widely used in the scientific community as a highly portable multi-platform car-racing simulator. It runs on Linux (all architectures, 32 and 64 bit, little and big endian), FreeBSD, OpenSolaris, MacOSX, and Windows (32 and 64 bit). It features many different cars, tracks, and opponents to race against. We can collect images for object detection as well as critical driving indicators from the game engine. These indicators include the speed of the car, the relative position of the ego-car to the center line of the road, and the distances to the cars in front.

The goal of the algorithm is to self-learn driving commands by interacting with the virtual environment. A deep reinforcement learning paradigm was used for this purpose, in which a deep convolutional neural network (DNN) is trained by reinforcing actions a that provide a positive reward signal r(s',a). The state s is represented by the current game image as seen in the simulator window. There are four possible actions: accelerate, decelerate, turn left, and turn right. The DNN computes a so-called Q-function, which predicts the optimal action a to be executed for a specific state s. In other words, the DNN calculates a Q-value for each state-action pair. The action with the highest Q-value will be executed, which moves the simulator environment to the next state s'. In this state, the executed action is evaluated by means of the reward signal r(s',a).

For example, if the car was able to accelerate without a collision, the related action which made this possible will be reinforced in the DNN; otherwise, it will be discouraged. The reinforcement is performed in the framework by retraining the DNN with the state-reward signals. Figure 7 shows the Caffe implementation for the deep reinforcement learning algorithm. The network layers have the same color-coding as in figure 6.

Figure 7: A Caffe-based deep convolutional neural network structure used for deep reinforcement learning
Machine learning can be defined as a set of algorithms that facilitate predictions based on past learning.

In a machine learning algorithm, the input data is organized as data points. Each data point consists of features that describe the represented data. For example, size and speed are features that can differentiate a car from a bicycle on the street. Both the size and the speed of a car are usually higher than those of a bicycle. The goal of the machine learning methodology is to convert the input data into a meaningful output, such as classifying the input data into car and non-car data points or objects. The input is usually written as vector $x$, composed from several data points. The output is written as $y$.

Two- or three-dimensional input data can be illustrated and viewed in a so-called feature space, where each data point in $x$ is plotted with respect to its features. Figure 8 (a) shows a simplified example of a two-dimensional feature space that describes the car and non-car objects.

A so-called learned mapping function or model, $h_\theta(x)$, gives the difference between the feature vectors (e.g., the classification into car and non-car data points). The structure of the model ranges from a simple linear function, such as the line dividing car and non-car objects in figure 8 (a), to a complex non-linear neural network. The goal of the learning methodology is to determine the values of the $\theta$-coefficients, which represent the parameters of the model from the available input data. The output of the mapping function is the algorithm’s prediction of what the input data describes.

Machine learning methods can be classified according to how the mapping function is learned (see figure 9, page 8). There are three possibilities:

- **Supervised learning.** The mapping function is calculated from training data pairs where the output $y$, known in advance, is given to the learning algorithm.
From logistic regression to self driving cars

- Figure 9: Classification of machine learning algorithms based on their training methodology

- Figure 10: Deep convolutional neural network trained to recognize cars in images

The so-called deep learning paradigm has revolutionized the machine learning field in recent years. Deep learning made a huge impact on the machine learning community by solving challenges that previously could not be tackled with traditional pattern recognition approaches (LeCun et al. 2015). The introduction of deep learning has dramatically improved the precision of systems designed for visual recognition, object detection, speech recognition, anomaly detection, or genomics. The key aspect of deep learning is that the features used to interpret the data are learned automatically from the training data instead of being manually crafted by an engineer.

Unsupervised learning. In this case there are no feature-label pairs available during the training phase, in contrast to supervised learning. The input to the learning algorithm consists only of unlabeled data points. The goal of this machine learning methodology is to deduce labels for the input features \( x \) directly from their distribution in the feature space.

Reinforcement (semi-supervised) learning. The training data has no labels in this case either, but the model is constructed to facilitate an interaction with its environment through a set of actions. The mapping function maps the state of the environment which is given by the input data to actions. A reward signal indicates the performance of an action on a certain state of the environment. The learning algorithm reinforces the action when the signal indicates a positive influence. The algorithm will discourage the specific action or state of the environment if a negative influence is recognized. The algorithm will discourage the specific action or state of the environment if a negative influence is recognized.

Architecturally, a deep learning system is made up from several layers of non-linear units, which can transform the raw input data into higher levels of abstraction. Each layer maps the output of the previous layer into a more complex representation that is suitable for regression or classification tasks. This learning is usually performed on a deep neural network that is trained by the use of a backpropagation algorithm. This algorithm iteratively adapts the parameters or weights of the
network in order to mimic the input training data. The network thus has learned a complex non-linear mapping function of the input data points by the end of the training.

Figure 10, page 8, shows a symbolic representation of a deep neural network that is trained to recognize cars in images. The input layer represents the raw input pixels. Hidden layer 1 usually mimics the presence or absence of edges in certain locations and orientations of the image. The second hidden layer models object parts using the edges calculated in the previous layer. The third hidden layer builds an abstract representation of the modeled objects, which, in our case, is the way a car is imaged. The output layer calculates the probability that a given image contains a car, based on the high level features of the third hidden layer.

Different network architectures result from the way that the units and layers of a neural network are distributed. The so-called perceptron is the simplest, consisting of a single output neuron. A large number of neural network flavors can be obtained by building on the perceptron. Each of these networks is more suited to a specific application than others. Figure 11 shows three of the most common neural network architectures out of the many that have been created in recent years.

A deep feed-forward neural network (figure 11a) is a structure in which the neurons between two neighboring layers are fully interconnected and the information flow is in one direction only, from the input to the output of the system. These networks are useful as general purpose classifiers and are used as the basis for all other types of deep neural systems.

The deep convolutional neural network (figure 11b) changed the way that visual perception methods are developed. Such networks are composed of alternate convolutional and pooling layers that learn object features automatically by generalization from the input data. These learned features are passed on to a fully interconnected feed-forward network for classification. This type of convolutional network is the basis of the car detection architecture shown in figure 10 and the use cases described in section 2.

While deep convolutional networks are crucial to visual recognition, deep recurrent neural networks (figure 11c) are essential for natural language processing. The information in such architecture is time-dependent due to the self-recursive connections between the neurons in the hidden layers. The output of the network can vary depending on the order in which data is fed into the network. For example, if the word cat is fed in before the word mouse, a certain output is obtained. Now, if the input order changes, the output order may change, too.

### Types of machine learning algorithm

Although deep neural networks are among the most often used solutions to complex machine learning challenges, there are various other types of machine learning algorithm available. Table 1 classifies them according to their nature (continuous or discrete) and training type (supervised or unsupervised).

Machine learning estimators can be classified roughly according to their output value or training methodology. The algorithm is classed as a regression estimator if the latter estimates a continuous value function \( y \in \mathbb{R} \) (i.e., a continuous output). The machine learning algorithm is called a classifier when its output is a discrete variable \( y \in \{0,1,\ldots,q\} \). The traffic sign detection and recognition system described in section 2.1 is an implementation of this type of algorithm.
Anomaly detection is one special application of unsupervised learning. The goal here is to identify outliers or anomalies in the data set. The outliers are defined as feature vectors that have different properties compared to the feature vectors commonly encountered in the application. In other words, they occupy a different position in the feature space.

Table 1 lists some popular machine learning algorithms. These are briefly explained below:

- **Linear regression** is a regression method used to fit a line, a plane, or a hyperplane to a dataset. The fitted model is a linear function which can be used to make predictions on the real value function y.

- **Logistic regression** is the discrete counterpart of the linear regression method, in which the predicted real value given by the mapping function is converted to a probability output that denotes membership of the input data point to a certain class.

- **Naïve Bayes classifiers** are a set of machine learning methods built on the basis of Bayes theorem which makes the assumption that each feature is independent of the other features.

- **Support vector machines (SVM)** are designed to calculate the separation between classes using so-called margins. The margins are computed to be as wide as possible in order to separate the classes as clearly as possible.

- **Ensemble methods**, such as decision trees, random forests, or AdaBoost, combine a set of base classifiers, sometimes called “weak” learners, with the purpose of obtaining a “strong” classifier.

- **Neural networks** are machine learning algorithms in which the regression or classification problem is solved by a set of interconnected units called neurons. In essence, a neural network tries to mimic the function of the human brain. Section 3.1 is dedicated solely to neural networks and deep learning.

- **k-means clustering** is a method used for grouping together features that have common properties, i.e., they are close to each other in the feature space. k-means iteratively groups common features into spherical clusters based on the given number of clusters to group.

- **Mean-shift** is also a data clustering technique, which is more general and robust with respect to outliers. As opposed to k-means, mean-shift requires only one tuning parameter (the search window size) and does not assume a spherical prior shape for the data clusters.

- **Principal components analysis (PCA)** is a data dimensionality reduction technique that transforms a set of possibly correlated features into a set of linearly uncorrelated variables named principal components. The principal components are arranged in order of variance. The first component has the highest variation; the second has the next variation below this, and so on.

<table>
<thead>
<tr>
<th>Supervised</th>
<th>Unsupervised</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Continuous (regression)</strong></td>
<td>Linear regression</td>
</tr>
<tr>
<td></td>
<td>Ensemble methods</td>
</tr>
<tr>
<td></td>
<td>k-means clustering</td>
</tr>
<tr>
<td></td>
<td>Mean shift</td>
</tr>
<tr>
<td></td>
<td>Principal components analysis (PCA)</td>
</tr>
<tr>
<td><strong>Discrete (classification)</strong></td>
<td>Logistic regression</td>
</tr>
<tr>
<td></td>
<td>Naïve Bayes</td>
</tr>
<tr>
<td></td>
<td>Support vector machines</td>
</tr>
<tr>
<td></td>
<td>Ensemble methods</td>
</tr>
<tr>
<td></td>
<td>Reinforcement learning</td>
</tr>
<tr>
<td></td>
<td>k-means clustering</td>
</tr>
<tr>
<td></td>
<td>Mean shift</td>
</tr>
</tbody>
</table>

Table 1: Types of machine learning algorithms

Deep learning has revolutionized machine learning systems and their capabilities, but it is not necessarily the most suitable approach for all tasks. It may be more appropriate to use traditional pattern recognition methods such as logistic regression, naïve Bayes, or k-means clustering for several other types of application. Criteria for selecting the right machine learning algorithm are therefore necessary. These criteria are described below.

The complexity of the problem is a straightforward criterion governing choice, which must fit the complexity of the method. This criterion can be translated into the number of parameters that the algorithm has to learn. As an example, the logistic regression algorithm learns two parameters for the mapping function $h_\theta(x)$ in Figure 8. A deep neural network might be required to learn millions of parameters to get at best similar results to those of the logistic regression method. Figure 12, page 11, shows an approximate distribution of machine learning algorithms ordered according to their complexity.
The math behind each algorithm is the basis for this empirical finding. The bias-variance tradeoff is one important aspect when choosing and building a machine learning system. Bias is the error produced by erroneous assumptions made by the learning method. It is directly related to the issue of underfitting. High bias algorithms fail to find the relevant relationships between the input features and the target labels. In contrast, variance is a measure of the sensitivity of the method to the random noise that is present in the input data. A high variance system can cause overfitting, where the algorithm models the random noise instead of the actual input features. In practice, a tradeoff between bias and variance must be found because these two quantities are proportional to each other. Another criterion that should be taken into account is the number of tuning parameters that a data engineer needs to tune when training a classifier.

Finally, the nature of the input data also needs to be considered. Linear separation of the data in the feature space is unusual in the real world. Arguably though, linearity can be assumed for some applications. An example of this is the classification of car and non-car objects based on their size and speed described at the beginning of section 3. This assumption is crucial in choosing an appropriate machine learning approach, since a linear classifier is faster and more effective for data that can be separated linearly compared to a non-linear classifier.

Functional safety considerations

Functional safety is part of the overall safety of a system. ISO 26262 “Road vehicles - Functional safety” describes the development of electrical and electronic (E/E) systems in road vehicles. A system is made safe by various activities or technical solutions. These so-called safety measures are reflected in the process activities that specify requirements, create architecture and design, and perform verification and validation. The avoidance of systematic failures is one aspect of ISO 26262. Human failures have been the systematic failures in traditionally engineered systems. Some clear examples of such failures are: requirements and test cases that are incomplete, significant aspects of the design that are forgotten, or verifications that fail to discover issues. The same is also true when using machine learning. Furthermore, the task to be learned and the corresponding test cases are all also described by humans. Systematic failures can still occur here. The development of machine learning models therefore requires the application of best practice or of an appropriate standard process. This alone is not enough. Safety measures are required in order to control systematic failures in the machine learning algorithms, given that parts of the development of system elements will be accomplished in future by means of such algorithms. These failures can be eliminated only if both can be guaranteed.

More attention has been given recently to safety in the context of machine learning, due to its increased use in autonomous driving systems. Amodei et al., 2016, discussed research problems related to accident risk and possible approaches to solving them. The code in traditional software systems has to meet specific requirements that are later checked by standardized tests. In machine learning, the computer can be thought of as taking over the task of “programming” the modules by means of the learning method. This “programming” represents learning the parameters or weights of the algorithm when considering the technical background presented in section 3. The learning procedure is very often stochastic, which means that no hard requirements can be defined. The machine learning component is therefore a black box system. As a result, it is difficult or even impossible to interpret the learned content, due to its high dimensionality and the enormous number of parameters.

Environmental sensors and the related processing play a decisive role that is beyond the requirements of functional safety, especially in the case of highly automated driving. The safety of the intended functionality (SOTIF) is concerned with the methods and measures used to ensure that safety-critical aspects of the intended functionality perform properly, taking sensor and processing algorithms into account. However, this problem has to be clarified for traditionally engineered systems and machine learning systems alike, and it is still the subject of ongoing discussions.

Analysis within a virtual simulator is one approach to disclosing such algorithms. We used this approach for...
From logistic regression to self driving cars

the experiments with self-learning systems that are presented in section 2.2. A theoretically infinite number of driving situations can be learned and evaluated in such a simulated environment before the machine learning system is deployed in a real-world car.

Lives are at stake now that machine learning has progressed from gaming and simulation into real-world automotive applications. As discussed, functional safety issues are becoming important as a result, and this also affects the scientific community. One consequence is research into approaches to benchmarking different machine learning and artificial intelligence algorithms in simulation. OpenAI Gym (Brockman et al. 2016) is one such simulator that is a toolkit for developing and comparing reinforcement learning algorithms.

Conclusions and outlook

The application of machine learning based functionalities to highly automated driving has been motivated by recent achievements. Initial prototypes have indeed produced promising results and have indicated the advantages when addressing the related complex problems. A significant number of challenges remain, however, even though machine learning can be suitable. It is necessary first of all to select the right neural network type for the given task. This selection is related to the applied learning methodology, the necessary preprocessing, and the quantity of training data. There is still discussion about the best way to decompose the overall driving task into smaller sub-tasks. Deep-learning technologies are capable of enabling end-to-end approaches without any need for decomposition, but this is currently considered to be less appropriate with regard to verification and validation capabilities. The machine-learning community needs to develop enhanced approaches, not least in order to address functional safety requirements, which are the foundation for successful industrialization of related functionalities.

Elektrobit is convinced that machine learning has the potential to reshape the future automotive software and system landscape, despite the challenges that remain. For this reason, two aspects of investigation have been started. The first is the application of machine-learning-based approaches as a solution to (selected subsets of) highly automated driving scenarios, such as the use cases mentioned above. The EB robinos reference architecture as well as the partnership with NVIDIA among other things contribute to the development environment. In the second, Elektrobit uses its expertise in the area of functional safety and industrialization of automotive software to bring these ideas and the products of its partners and customers to life.

Bibliography


From logistic regression to self driving cars

Author: Sorin M. Grigorescu
Assoc. Prof. Sorin Mihai Grigorescu received his Dipl.-Eng. degree in Control Engineering and Computer Science from the Transilvania University of Brasov, Romania, in 2006 and his Ph.D. in Robotics from the University of Bremen, Germany, in 2010. Between 2006 and 2010 he was a member of the Institute of Automation at the University of Bremen. Since June 2010, he has been affiliated with the Department of Automation at Transilvania University of Brasov, where he has been leading the Robust Vision and Control Laboratory. Since June 2013, he has also been affiliated with Elektrobit Automotive Romania, where he is the team manager of the Navigation Department. Sorin M. Grigorescu was an exchange researcher at several institutes, such as the Intelligent Autonomous Systems Group (Technical University Munich), the Korea Advanced Institute of Science and Technology (KAIST), or the Robotic Intelligence Lab in University Jaume I (Spain). Sorin was the winner of the EB innovation award 2013 for his work on machine learning algorithms used to build next generation adaptable software for cars.

Author: Markus Glaab
Markus Glaab is an expert for automotive software architectures at EB Automotive Consulting. In 2010, he received his M.Sc. degree in Computer Science from the University of Applied Sciences Darmstadt, Germany, while gaining professional experience as a software developer in the area of Car2X communication. In 2010, he also started to do research at both the In-Car-Multimedia Labs at his home university, where he worked on future service delivery platforms for vehicle-to-backend architectures, and the Centre for Security, Communications and Network Research at Plymouth University, UK. Markus has been with EB since 2016 and works on future automotive E/E architectures and the integration of related technologies such as machine learning.

Author: André Roßbach
André Roßbach is a senior expert for functional safety at EB Automotive Consulting. In 2003, he received his degree in Business Information Technology from the University of Applied Sciences Hof. He then gained experience as a software developer in the biometric and distributed database development area. André has been with EB since 2004 and has developed software for medical systems, navigation software, and driver assistance systems. Currently, his focus is on functional safety, agile development, and machine learning.
From logistic regression to self-driving cars

About EB Automotive
Elektrobit (EB) is an award-winning and visionary global supplier of embedded and connected software solutions and services for the automotive industry. A leader in automotive software with over 25 years serving the industry, EB’s software powers over 70 million vehicles and offers flexible, innovative solutions for connected car infrastructure, human machine interface (HMI) technologies, navigation, driver assistance, electronic control units (ECUs), and software engineering services. EB is a wholly owned subsidiary of Continental.