TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Maneuver prediction using vehicle sensor data

Authors	Anna Kuvakina, Harshit Chopra, Nils Sturma
Mentors	Nico Epple M.Sc. (BMW AG),
	Dr. Benny Kneissl (BMW AG)
Co-Mentor	Laure Vuaille M.Sc. (Department of Mathematics)
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

July 2019

Abstract

Advanced driver assistance systems (ADAS) are one of the dominant topics in road safety in recent years. In order to describe the impact in individual driving situations, a suitable classification of the traffic situation is essential. In this work, we will focus on the intention of the driver and develop an approach for lane change prediction based on naturalistic driving studies (NDS) conducted by BMW. We use data from test vehicles and predict the probability of a lane change through the positioning and movement of the vehicles in the surrounding. How the driver behaves in the vehicle and the vehicles he can see in the rear-view mirror, we deliberately neglect here. In order to take cultural and infrastructural influences into account, we implement our model on German data only. In a preliminary step, we develop a naive algorithm to identify lane changes in a Big Data environment. With this automatically generated ground truth, we create labels and randomly validate them with videos. Due to the complex patterns with spatial and temporal dependencies, our model uses time series as input and predicts through engineered features whether a lane change will follow or not. We use different machine learning techniques like Random Forests, Support Vector Machines (SVM) and Recurrent Neural Networks (RNN). With Random Forests and SVM, we create baseline models and apply feature selection techniques. Then, we improve our efficiency implementing a RNN model.

Contents

1 Introduction

The range of functionality of advanced driver assistance systems (ADAS) and their number has increased drastically in recent years. Due to the demand for highly automated driving (Level 2 to Level 5, SAE-J3016), this trend will persist in the upcoming years. Such systems are also meant to support the driver and particularly taking control in potentially dangerous situations. These situations often correlate with active maneuver like lane changes. To provide ideal support for the drivers and assess the interaction of ADAS within traffic scenarios, it is essential to predict the driver's intentions as a part of the situation definition. One possibility to tackle this issue is conducting driving studies with measurement equipped probe vehicles. Naturalistic Driving Studies (NDS) collect real-world traffic data of minimally influenced driving behavior and take place in various countries.

Predicting the intention of a driver before he starts his actual maneuver is a big challenge. Individual persons behave differently, have a different perception of their environment, and this also deviates from the reality recorded in the measurement data. Besides prediction, these edge cases of false classification are also interesting to explore. They reflect the differences in human driving behavior and the deviation in the perception and interpretation of the environment.

2 Data set

In this project, we use real world traffic data collected in Germany. The first part of the data is the *signal data*, and it consists of continuous time series resampled at a frequency of 10 Hz. Its structure is described in *table* 1.

Feature	**Description**	**Type**
vel_x	Longitudinal velocity (km/h)	Quantitative
acc_x	Longitudinal acceleration (m/s^2)	Quantitative
acc_y	Lateral acceleration (m/s^2)	Quantitative
heading	Compass - direction of driving $(0 = \text{north}, 180 = \text{south})$	Quantitative
dist_line_r	Distance from outer edge of right front wheel to the lane boundary (cm)	Quantitative
dist_line_l	Distance from outer edge of left front wheel to the lane boundary (cm)	Quantitative
road_lanecount	Count of the lanes (in same driving direction)	Ordinal
road_angle	Angle of road	Quantitative
winker	Winker off, left on, right on, both on	Nominal
steering_wheel_angle	Angle of steering wheel	Quantitative
lane_change_l_warning	Warning whether there is an object in the left blind spot or not	Dichotomous
lane_change_r_warning	Warning whether there is an object in the right blind spot or not	Dichotomous
boundary_line_l	Type of left boundary line of lane	Nominal
boundary_line_r	Type of right boundary line of lane	Nominal

Table 1: Features in signal data

The second part of the data is *object data*. It is also resampled at 10 Hz. In the literature, the vehicle for which the maneuver prediction is done is usually called *Ego*. At each time stamp, there can be a variable amount of objects surrounding the Ego vehicle. The data of these objects is obtained and classified. For example, an object can be a person, a car, a truck or a bicycle. The features, as mentioned in *table* 2, are collected for each object. Some features like the position are measured relative to the Ego vehicle.

Feature	**Description**	**Type**
classificationType	Indicates Type of detected object	Nominal
statusMovement	Indicates direction of movement relative to Ego	Nominal
drivingTubeMapping	Indicates in which lane the object is located relative to Ego	Nominal
position	Longitudinal distance relative to Ego (m)	Quantitative
positionY	Lateral distance relative to Ego (m)	Quantitative
vel_x	Absolute longitudinal velocity (m/s)	Quantitative
vel_y	Absolute lateral velocity (m/s)	Quantitative
acc_x	Absolute longitudinal acceleration (m/s^2)	Quantitative
acc_y	Absolute lateral acceleration (m/s^2)	Quantitative
yawAngle	Angle with respect to yaw axis	Quantitative
yawSpeed	Rate of change of yaw angle	Quantitative
height	Height of object (m)	Quantitative
length	Length of object (m)	Quantitative
width	Width of object (m)	Quantitative

Table 2: Features in object data

3 Preprocessing

3.1 Data acquisition

In order to predict lane change maneuvers, the data has to be annotated with labels first. Due to the large data size, it is not possible to manually label all lane changes by detecting them in the corresponding driving videos. Instead, we develop a rule based algorithm which automatically finds these events in a given time series of data. The result of this algorithm, we want to use as our ground truth to build a model which predicts lane change maneuvers.

3.1.1 Labeling

In this project, we define a lane change event as the time between the first front wheel crosses the line until the second front wheel crosses it.

We identify lane changes in multiple stages. Precisely, we first identify overline situations which are all situations where one wheel crosses a line. Then, a lane change into the left lane is a situation where first the left wheel crosses the line to the left and then (in a certain time window) the right wheel crosses the line to the left as well. In the same manner, we can find a lane change into the right lane. In order to find overline situations in a given sequence of data, we use the features *dist_line_r* and *dist_line_l*. These features are the distance from the outer edge of the front wheel to the right and left line respectively. Each sensor has a range of 152 to −100 cm to the lane. Negative values mean that the corresponding wheel is currently over the line. *Figure 1* shows that if there is a sign change in the distance to the line, which is not caused by a jump then we consider it to be an overline situation. In previous work, evaluating real world traffic BMW found that over 95% of lane changes take less than 8 seconds. Therefore, we set the maximal length of the time window during which the second wheel should cross the line after the first wheel crosses it to 8 seconds.

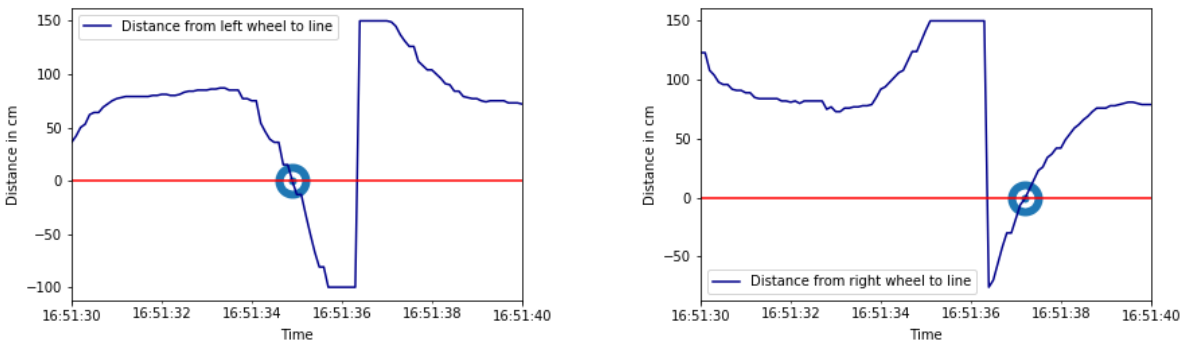


Figure 1: Lane change maneuver into left lane. First, the left wheel crosses the line. Second, the right wheel crosses the line.

After developing the labeling algorithm based on overline situations, we test it on 6 hours of randomly selected driving data. We compare our labels determined by the algorithm

with videos of the corresponding drives. In this testing data, all labeled events are indeed lane changes. *Figure 2* shows an example of the result of the algorithm. As it is shown, the algorithm is able to differentiate between left and right lane changes. Nevertheless, it does not identify all lane changes in the data. For example, this is the case when the lines on the streets are missing or the sensor which measures the distance to the line does not work correctly.

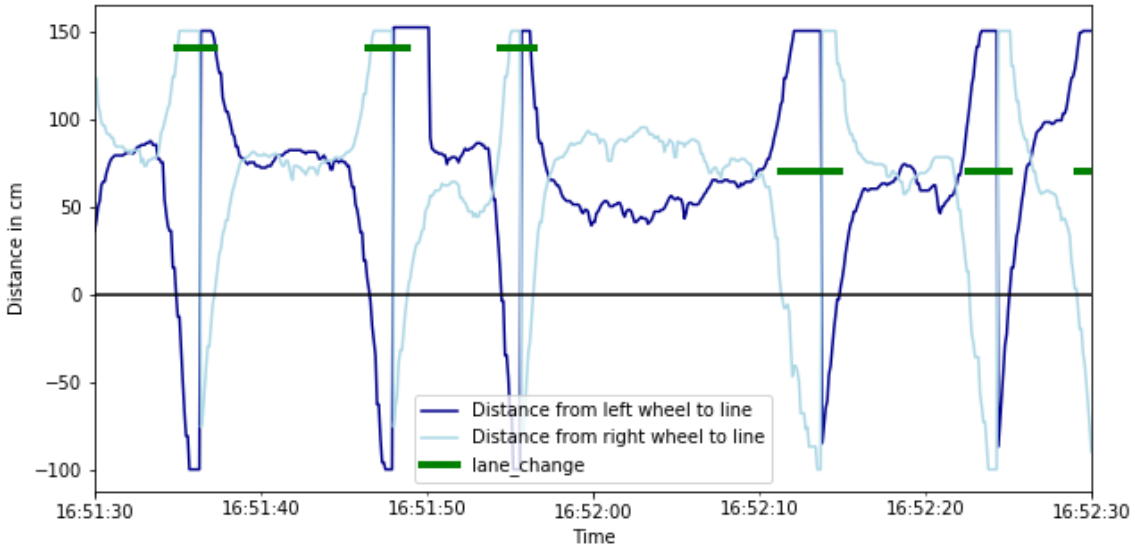


Figure 2: Distances to the lines from both wheels. Upper green bars: Lane change into left lane. Lower green bars: Lane change into right lane.

In addition, we test our algorithm allowing different lengths of time windows between crossing the line with the first wheel until crossing the line with the second wheel. *Figure 3* shows that we get indeed over 95% of the lane changes in our testing data when we allow a maximum of 8 seconds for lane change duration.

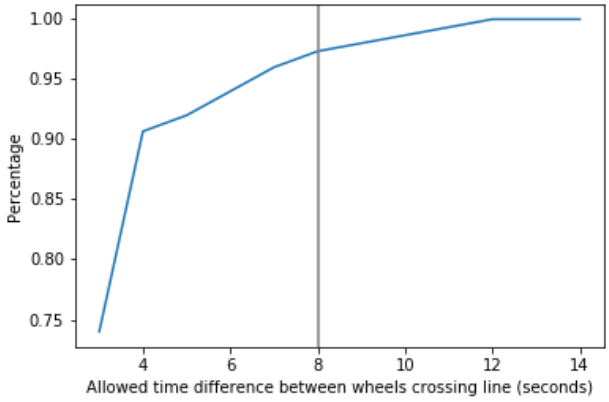


Figure 3: Cumulative percentage of lane change duration.

3.1.2 Prediction horizon

Our maneuver prediction method has to access whether a lane change is following in the near future or not. The time difference between the model prediction and the actual start of lane change event is defined as prediction horizon. As described in [8], *figure 4* shows the events which happen sequentially in a lane change situation.

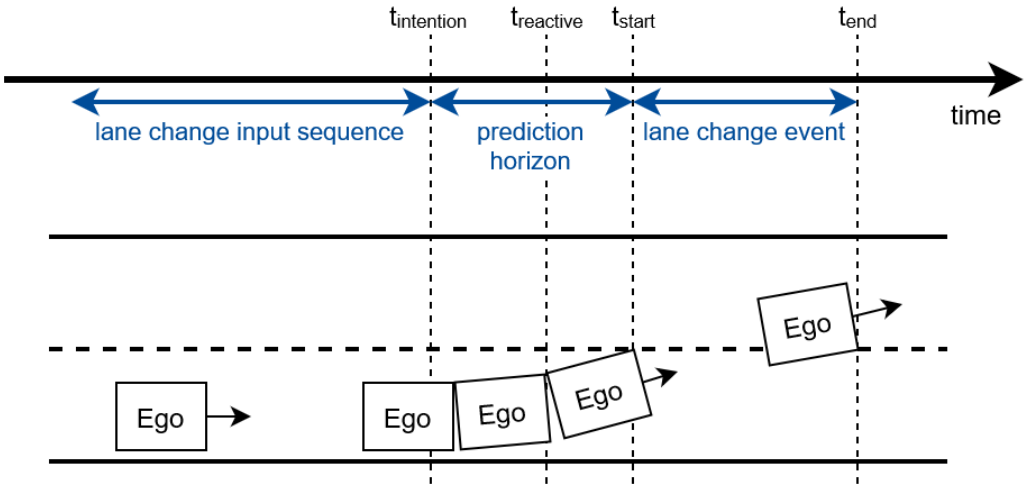


Figure 4: Prediction horizon of a lane change to the left.

The points $t_{intention}$ until t_{end} are defined in the following way:

- $t_{intention}$: Intention/ decision of the driver to make a lane change.
- $t_{reactive}$: Lane change maneuver has started, vehicle is moving towards lane boundary.
- t_{start}: Start of lane change event, first wheel crosses line.
- t_{end}: End of lane change event, second wheel crosses line.

Clearly, the lane change has not finished at t_{end}, but for our goal it is not necessary to consider the time after t_{end}. In this project, we aim to predict the intention of the driver. Therefore, the prediction horizon is the time between $t_{intention}$ and t_{start}.

To get a reasonable value for $t_{intention}$, we randomly take 500 lane change events from the German highway data. *Figure* 5 shows the distribution when drivers turn on the winker directly before a lane change. Out of the 500 situations, the winker is turned on in 73% of the cases. Since the winker is a good point to determine when a driver decides on a lane change, we obtain the median value of the winker start for these cases. The median time of turning on is 1.5 seconds before t_{start}.

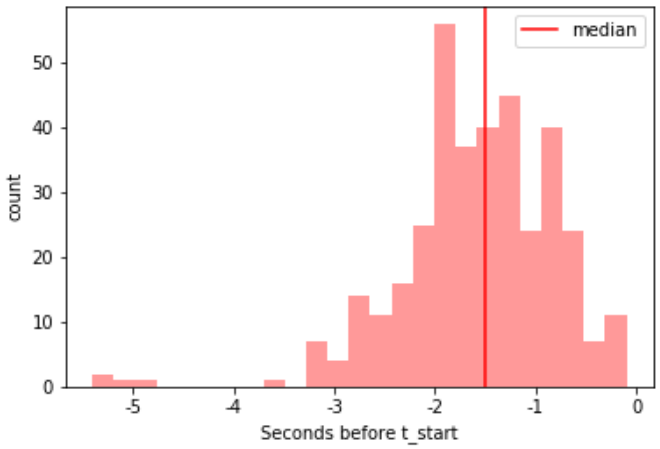


Figure 5: Histogram - time winker is switched on directly before a lane change event happens.

Therefore, we determine the prediction horizon as 1.5 seconds. Any model predicting the intention of a driver should detect an upcoming maneuver (in our case a lane change) out of the situation and the surrounding objects. Hence, our model should depend only on features that influence a lane change maneuver and not on those that help to identify one. As a consequence, we do not use the features *winker*, *dist_line_r*, *dist_line_l*, *steering_wheel_angle* and *acc_y* for training our models.

3.1.3 Final input data

After developing the algorithm to automatically label lane changes and determining the prediction horizon, we extract data from the BMW cluster. As an input for the models we use small time series of data. For lane change samples these series end at $t_{intention}$ (*figure 4*). Clearly, for non-lane change samples we use sequences where no lane change maneuver is happening afterwards. Due to computational reasons and the amount of data, we decide to use sequences of 5 seconds as input samples. In addition, we resample the input sequences to a frequency of 2 Hz. Extracting data from the cluster is an iterative process as shown in *figure* 6.

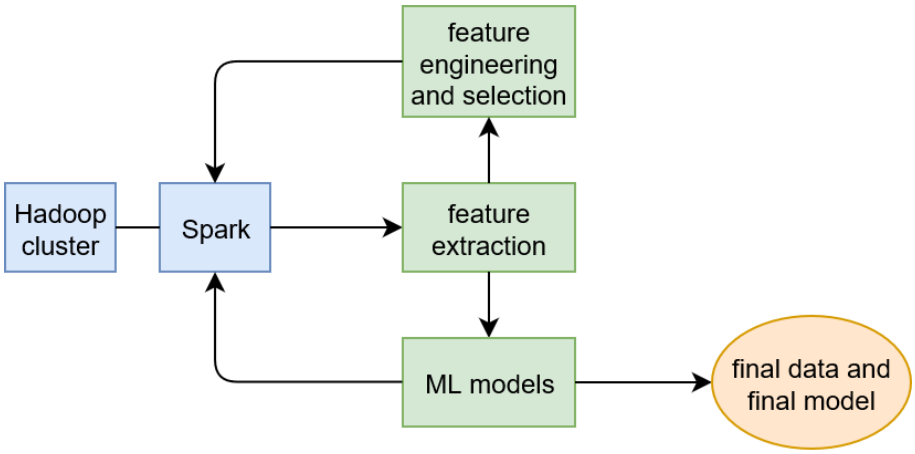


Figure 6: Data processing chain.

Motivated by the size of the data, it is stored in a distributed environment (Hadoop) and processed in a Spark framework. We work in a closed loop where we first extract a small amount of data which we explore. In the next step, we perform feature engineering and selection on that data. Then, we come back to the Spark framework and extract these features on a large scale. After a few iterations we start training machine learning models. Next, we evaluate the results and go back to the Spark framework to extract new features if needed. At the end of these iterations, we obtain the final data set which is described in *table* 3. As it is realistic to have more non lane change samples than lane change samples, the data is unbalanced. To create a realistic test set, we take randomly 5000 samples from the training data.

	Train	**Test**
lane change samples	13416	887
non lane change samples	68393	4113

Table 3: Size of data used in this project.

To ensure that we obtain only highway data, we check extra conditions. We just use samples where the mean velocity of Ego is higher than 60 km/h and in at least one of the time stamps of the input sequence, there is a *road_lanecount* value higher than or equal to 2. This means that the Ego is driving on a street where a lane change is possible. In the next two sections, we describe how we engineer new features and further clean the data.

3.2 Feature extraction

3.2.1 Surrounding vehicles and their features

In the object data described in *table 2*, we have data for all objects in front of the vehicle for each time stamp. We are interested in the objects in the direct neighborhood of the Ego vehicle. Therefore, we consider surrounding vehicles that we call E1, E2, E3 and E4 [10] as shown in *figure 7*. These objects drive in the same, left or right lane compared to Ego within the sensor range. They are the closest vehicles in front of Ego, drive in the

same direction as Ego and have a suitable object type (e.g. cars, motorbikes, trucks). For an example: To find E1, we obtain the closest vehicle to Ego that drives in front of it in the same lane and in the same direction. For each of these extracted objects, we consider the following features:

1. relative distance from Ego
2. absolute speed of the object
3. absolute acceleration of the object

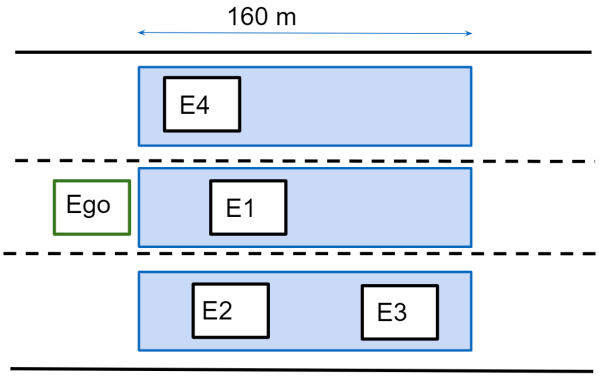


Figure 7: Objects that surround Ego vehicle.

We evaluate the correctness of extracted objects and their features by comparing them to objects in videos. That means, we take randomly some driving videos and validate that the objects observed in the videos fit to our extracted features. As it always fits together, we conclude that our approach for obtaining surrounding vehicles from the object data is correct.

3.2.2 Feature engineering

Moreover, we compute new continuous and categorical features for these four vehicles from the object data. To train a model for lane change predictions, *Salvucci et al. [9]* suggest to use the following input features: inverse time to collision and time headway. Hence, we calculate these between Ego and E1, E2, E4, as well as between E2 and E3. For example, inverse time to collision and time headway between Ego and E1 is shown in *equation 1* and *equation 2* respectively.

$$inv\_ttc\_Ego\_E1 = \frac{speed\_Ego - speed\_E1}{distance\_Ego\_E1} \tag{1}$$

$$thw\_Ego\_E1 = \frac{distance\_Ego\_E1}{speed\_Ego} \tag{2}$$

Exemplary, the distribution of time headway between Ego and E1 is shown in *figure 8*. Furthermore, we add two categorical features to our data set which are the total amount of objects in the right and left lanes regarding the Ego vehicle.

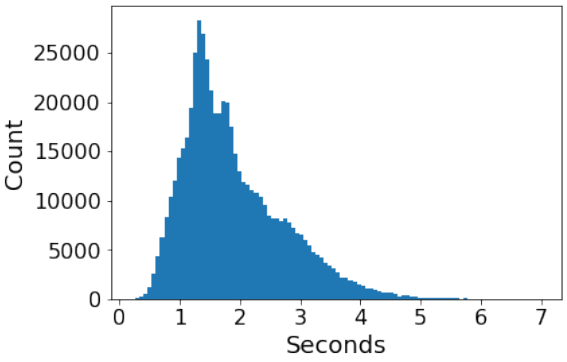


Figure 8: Time headway for Ego and E1 for all data

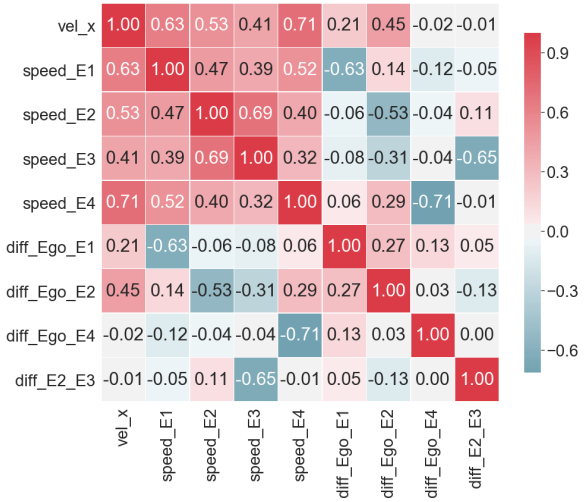


Figure 9: Correlation matrix

We analyze correlation between features and see that the absolute speed of surrounding vehicles is correlated with the speed of the Ego itself. Therefore, we decide not to use the absolute speed of the objects but the relative speed to the Ego. In *figure 9*, one can see that the speed difference is, in general, less correlated to the velocity of the Ego than the absolute speed itself. The whole correlation matrix for all features can be found in *figure 21* in the Appendix.

3.3 Data cleansing

In this section, we explain how we ensure that the data is clean, how we handle missing data and what scaling methods we apply. For this data set, cleaning data particularly means dealing with missing values and filling missed point observations with interpolations. The signal data is already interpolated and cleaned such that we use it without the application of any interpolation strategies.

3.3.1 Handling missing objects

The necessary objects as described above are missing for particular time stamps in some cases. The key reasons for this is either there was no vehicle for that particular position present in the vicinity of the Ego vehicle in that time stamp. Or, for technical reasons, there is no information about certain situations. We drop input sequences where more than 50% of the information is missing.

The initial data is recorded at 10 Hz frequency. Since it is reasonable to assume that the raw values do not change much in successive entries (with a difference of 0.1s), we fill the missing entries with the values from the previous time stamp in the same input sequence.

A limit of 3 (0.3 s) was applied on the forward fill. The other values in the sequence that are still missing, are filled later with the mean of that particular feature. In order to differentiate these values from the actual values, we create new binary features for each of the surrounding objects. These binary features have a value of one if an object is found and zero if no object is found for that particular time stamp.

3.3.2 Scaling methods

We scale the features to make the model perform better and learn the relations in the data quicker. The values needed to scale the data are calculated on the train set and then both, train and test set are scaled accordingly. For continuous features, we use the following two approaches to test which one performs better for our input data.

- **Standard Scaling**

  $$x_{new} = (x - \mu)/\sigma$$

  where μ is the mean across that feature and σ is the standard deviation.

- **Min-max scaling**

  $$x_{new} = (x - x_{min})/(x_{max} - x_{min})$$

  where x_{max} and x_{min} are the minimum and maximum values for that particular feature.

The nominal features with more than two possible categories are one-hot-encoded. That means that we create a new binary feature for each category. The number of new features equal the number of unique categories in the initial feature. Some of the features that are one-hot encoded are *lane_change_l_warning* and *boundary_line_r*. We do not one-hot-encode nominal features like number of vehicles in a lane as the value has physical significance. These are scaled using the above mentioned techniques.

The final set of features is shown in *table 8* in the Appendix.

4 Statistical data exploration

We statistically explore the input data for our models. Therefore, we create data exploration plots to observe general patterns and the effect of features on the possibility of a lane change. Precisely, we compare the 5 second input sequences labeled as lane change with sequences not labeled as lane change. Our aim is only to predict if a lane change is upcoming or not. Nevertheless, it makes sense to explore the data by distinguishing between lane changes to the left and lane changes to the right. In the following, we show examples of the most interesting features. The rest can be found in *figures 22* and *23* in the Appendix.

4.1 Continuous features

We see that lane changes to the left have an increasing average acceleration by the Ego vehicle. This is shown in *figure 10*. It complies with the fact that drivers often change to the left lane to either overtake or to increase their speed. In comparison, when the vehicle changes to the right or stays in the lane, the acceleration does not increase.
Figure 11 shows the mean headway time between Ego and E1. On average, we see a decrease in the time headway for left lane changes. Since changes to left are often because Ego is faster and needs to overtake E1, this observation is expected. On the other hand, if the Ego vehicle does a lane change to the right, on average we see an increase in the time headway. This behaviour we expect as well, since in this case E1 is faster than the Ego vehicle itself. That is why Ego changes from the left into the right, a slower lane. Lastly, the time headway on average stays constant if the sequence is labeled as no lane change.

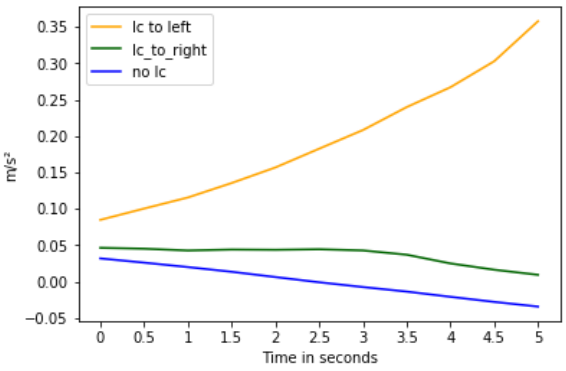


Figure 10: Mean acceleration of Ego vehicle.

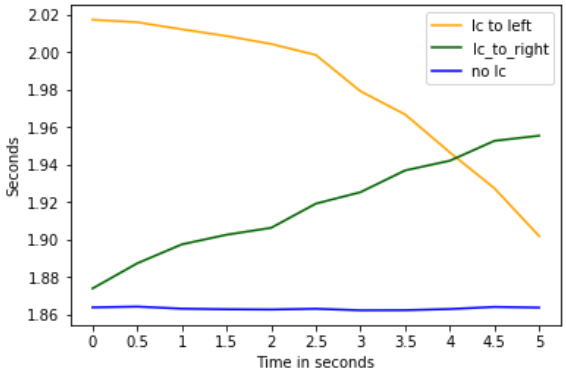


Figure 11: Mean time headway between Ego and E1.

Figure 12 shows the inverse time to collision between Ego and E2. For lane changes to the right, we see on average a significant decrease at the end of the input sequence. This means the time to collision values are increasing. A potential explanation for this behaviour is more difficult. It could be that the Ego gets slower and therefore turns to the next lane right. On the other hand, we analyze that for lane changes to the left and no lane changes the time to collision between the Ego vehicle and E2 is on average very

similar. We conclude that a lane change to the left does not depend that much on E2. The mean values of a feature over the different labels give the sense that the data is highly separable. *Figure 13* also shows the standard deviation of each of these average values in the case for the inverse time to collision between Ego and E2. From that we see that the values highly overlap. This trend was observed for most of the features.

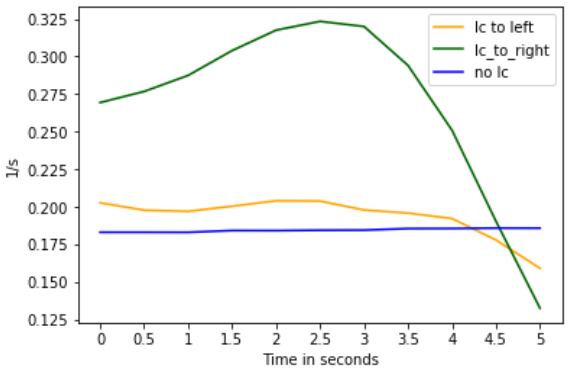


Figure 12: Mean inverse time to collision between Ego and E2.

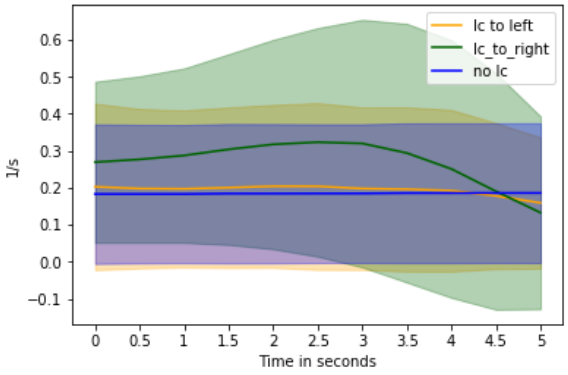


Figure 13: Mean inverse time to collision between Ego and E2 with std. deviations.

4.2 Categorical features

Next, we explore the categorical features. Here, only the last time stamp (time corresponding to $t_{intention}$, see *figure 4*) in each input sequence for certain categorical features is taken into consideration. We evaluate the effect on the probability of a lane change dependent on different categories.

Figure 14 shows the empirical probability of a lane change conditioned on the existence of the surrounding vehicles E2, E3 and E4. We see that the empirical probability of a lane change is reduced by the existence of E2 and E3. We expected this, since Ego would have less scope to change its lane if there is more surrounding traffic. In the case of E4, we see that the empirical probability of a lane is increased by its existence. This seemingly anomalous behaviour of E4 is probably because of another reason. The signal data consists of the number of lanes on the road but does not have information about which lane is the Ego vehicle currently driving in. The existence of E4 leverages the probability that a left lane exists to which the Ego vehicle can go.

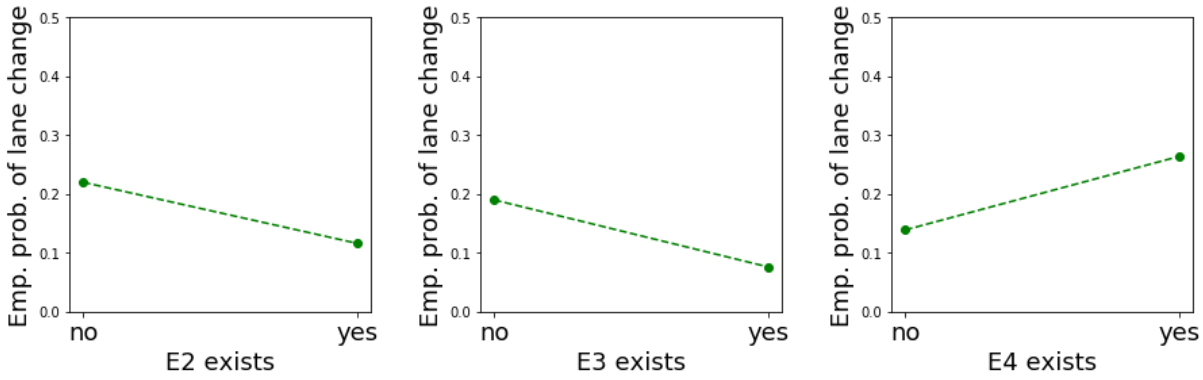


Figure 14: Probability of lane change conditioned on the existence of different vehicles.

We use the same method as described above to inspect other categorical variables. In *figure 15*, the first plot shows the effect of the type of boundary line on the empirical probability of lane changes. Category 2 leads to a very high probability of a lane change but occurs rarely. Other than that category 3 leads to a higher probability of lane change and occurs often. In the second plot, we see that probability of lane changes reduces if we have the lane change warning for the right activated (category 1), i.e there is some object in the right blind spot area of the Ego. The probability is not zero because you can still change your lane in the other direction and these values are collected 1.5 seconds before the start of lane changes. The object may have moved out of the blind spot region by that time. Category 2 we observe very rarely again. The last plot shows that the lane change probability decreases as number of vehicles around the Ego increase (here the count of vehicles in right lane is shown). This is also expected, because more traffic makes it difficult for the driver to change lanes.

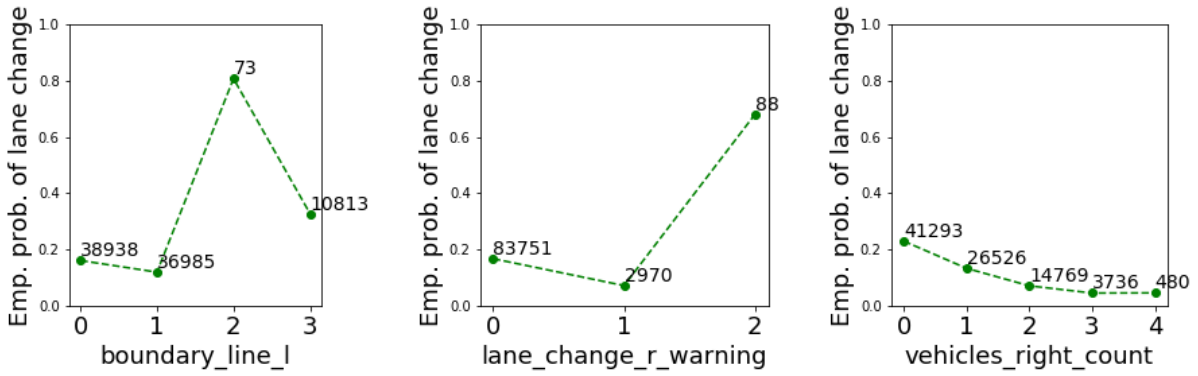


Figure 15: Probability of lane change conditioned on different categories. Numbers are counts on how often the category occurs.

5 Algorithms

In this section, we explain the performance metrics that we choose and the key reasons behind that. We describe the different machine learning techniques that are used for creating our maneuver predicting method. Random Forests and Support Vector Machines (SVMs) are frequently used methods for classification tasks. Our intention is to get a relatively simple baseline model using a Random Forest. Then, we use SVM where we apply feature selection methods in order to find out which features are most important for predicting lane changes. We improve the results by using Recurrent Neural Networks. Such networks are designed to handle time series input data. For fundamental knowledge of these models, we refer to [5], [1] and [6].

5.1 Metrics used to compare algorithms

The prediction of a model can be True Positive (TP), False Positive (FP), False Negative (FN) or True Negative (TP). TP represents the instances, where lane changes are correctly classified. FP describes the events that are wrongly classified as lane changes. Similarly, FN refers to lane changes that are wrongly classified as non lane changes. In the following equations, we define the performance measures that are used later for our comparisons [3].

$$precision = \frac{TP\_count}{TP\_count + FP\_count} \tag{3}$$

$$recall = \frac{TP\_count}{TP\_count + FN\_count} \tag{4}$$

$$F_1 = \frac{2 * precision * recall}{precision + recall} = \frac{TP\_count}{TP\_count + \frac{FN\_count + FP\_count}{2}} \tag{5}$$

Precision is the fraction of relevant instances over all retrieved instances as shown in *equation* 3. Recall is the fraction of retrieved relevant instances over all relevant instances and is defined in *equation* 4. In our case, a perfect precision score of 1.0 means that every obtained lane change is indeed a lane change, however, it does not necessarily mean that all lane changes are found. The highest recall score of 1.0 proposes that all lane changes are obtained, but does not say anything about how many sequences are incorrectly labeled as a lane change.

The combination of these two metrics is called F_1 score, which is the harmonic mean of precision and recall as given in *equation* 5. This is a weighted mean where low value weights are increased. Hence, F_1 gives equal weights to both precision and recall. Therefore, to achieve a decent F_1 score, both precision and recall should be high [3]. We use F_1 score to compare our models since classes in our data set are imbalanced. We also consider the accuracy score as an additional information.

5.2 Random Forest

A Random Forest is an ensemble of decision trees [3]. The advantage of tree based models is that there is no need to scale the quantitative features nor to one-hot encode the categorical features. In addition, these models are not sensitive to correlated features as explained in [6]. Therefore, we can just use all features. Random forests introduce additional randomness to avoid overfitting by searching not for the best feature when splitting a node, but searching for the best feature from random subset of features. Since our input samples are sequences, we process them before feeding to a Random Forest. For each input sequence, we extract the following features:

- all features corresponding to last time stamp of sequence
- all features corresponding to first time stamp of sequence
- the difference between the first and the last time stamp (again for all features)

We do not perform a systematic hyper parameter search since we only want to get a benchmark model. Instead, we manually test some combinations of parameters until we have a reasonable benchmark.

5.3 Support Vector Machine

An SVM is a powerful and versatile machine learning model, especially used to classify complex, but not large, datasets [3]. A Linear SVM classifier separates data into two data classes linearly. The separation line should not come close to the instances, but stay as far away as possible.

Usually a linear SVM perfoms well in classification tasks and linear kernel is a default one [7]. In our case, the two classes are not linearly separable, hence, we apply the kernel trick to the data. We observe that, a commonly used for SVM [3], Radial Basis Function (RBF) kernel performs better on our data set than a linear kernel. We transform the input sequences in the same way as we did it for Random Forests. Additionally computed feature differences are scaled in the same way as the original features, and our final data set consists of 90 features. We also use balanced class weights to make the weights inversely proportional to class frequencies. Next, we do a grid search with 10-fold cross validation with all the features to find the best hyper parameters. We obtain the best results for gamma of 0.01 and penalty parameter for the error of 10.

- **Gamma** - This regularization parameter defines the influence of each training sample that is selected as a support vector. Increasing gamma results in more irregular decision boundary that wiggles around individual instances. Hence, the model fits better the training samples. With smaller values of gamma, decision boundary becomes smoother.
- **Penalty parameter** - This parameter represents a trade off between correct classification of training samples and maximization of the margin of the decision function. Smaller penalty parameter leads to a larger margin. Hence, the decision function will be simpler at the cost of lower accuracy.

The number of features is very large, so we use two feature selection methods to find the most relevant features using the best hyper parameters from above. We cannot do a grid search for hyper parameters for every feature combination because it is computationally very expensive.

5.3.1 Forward feature selection

In [4], Guyon et al. suggest to use forward feature selection method to find features that perform best. The idea of this method is to train different models with every single feature and iteratively continue with the best feature from the last round. The procedure continues until all features are ranked or the model's performance stops increasing. The best features are shown in *table* 4. Prefix *diff* represents the difference between first and last time stamps. Suffix *start* in some feature names refers to a feature corresponding to the first time stamp.

Features	F_1 **score**
vehicles_right_count	0.3659
all above & boundary_line_l_3	0.3823
all above & vel_x	0.4018
all above & acc_x	0.4406
all above & diff_vel_Ego_E2	0.4863
all above & boundary_line_l_3_start	0.4977
all above & inv_ttc_Ego_E1	0.5107
all above & diff_vel_Ego_E2_start	0.5239
all above & thw_Ego_E1	0.5289
all above & acc_x_E1	0.5385
all above & distance_E1_start	0.5388
all above & diff_inv_ttc_Ego_E4	0.5456
all above & inv_ttc_Ego_E2	0.5474
all above & E3_exists	0.5533
all above & diff_thw_E2_E3	0.5571

Table 4: Best features obtained from forward feature selection method. Each next iteration includes the features from above and a new feature.

Number of vehicles in the right lane is the best feature, and its impact to a lane change we already saw in the data exploration, shown in *figure* 15. Adding boundary line type further increases the performance, which indicates if a lane change is possible at all. The next best features are the velocity and acceleration of the Ego vehicle. If one trains an SVM model with 15 features from *table* 4, then the F_1 score of slightly more than 55% can be achieved.

5.3.2 Backward feature elimination

In backward feature elimination, one trains a model with all the features and then keeps removing the least promising one until the performance stops increasing [4]. First, several features representing boundary type are eliminated. These features can be redundant

since we observe that feature boundary type left and right usually have the same values. Next, road angle and boolean variable if E3 exists are removed. Further feature reduction results in slight performance drop, hence, the algorithm terminates. We conclude that the dropped features are not necessary to predict lane changes, and we achieve a similar F_1 score of 56% as in the forward feature selection method.

Both feature selection algorithms improve kernel SVM performance by almost similar amounts. To train the best SVM model, we use the features remaining after the backward elimination process.

5.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. Such models have an internal state (memory) which allows them to process a series of inputs where the output of one sample is influenced by the previous sample [1].

For our project, RNNs are selected as the input is a time sequential representation of the kinematics and surrounding of the Ego vehicle. The input is reshaped such that each value consists of a 2-dimensional matrix with the features as columns and each row representing a particular timestamp. Thus, each final input sample has the following shape:

$$(number\_of\_timestamps * features)$$

We use Long Short Term Memory (LSTM) based RNNs for this purpose. A LSTM unit consists of an input gate, output gate and a forget gate. The key reason for using such a model is that they prevent the problem of vanishing and exploding gradient. In addition, we use Gated Recurrent Units (GRU) as they are computationally less extensive and perform very well on smaller time series [11]. The networks are implemented using the deep learning library Keras [2].

5.4.1 Model characteristics and hyperparameter tuning

We create a model with 4 layers: the first two are recurrent layers. This is followed by two dense layers with the latter one having size equal to the number of outputs. Some of the other features used in the network include:

- **Input Weight Sampling** - Since our dataset is highly skewed (almost 83% of the samples are non lane changes) to represent a real world scenario, we try weight sampling the inputs to give more importance to the lane change samples.
- **Dropout** - Dropout involves random switching off certain neurons in a particular layer during training. This prevents the model from overfitting on the training dataset. We use both normal and recurrent dropout for our model.
- **Weight regularization** - L2 regularization limits the absolute values of the weights. This technique also prevents overfitting, and we use it since our training and validation accuracy initially had a huge margin.

- **Reduce learning rate** - We use Keras callback ReduceLROnPlateau to reduce the learning rate whenever the validation loss stopps decreasing for a given number of epochs. Reduction in learning rate helps to fine tune the model when it starts to fluctuate between two sub-optimal values. We implement ReduceLROnPlateau instead of weight decay as it depends on feedback from the network instead of a fixed rule.

Finally, we train two RNNs, one with LSTM and the other with GRU as the recurrent layers. The final structure of the models is shown below:

Model	Recurrent 1	Recurrent 2	Dense 1	Dense 2	Learning rate
GRU	128	64	32	1	0.003
LSTM	128	128	32	1	0.005

Table 5: Structures of the best model after hyperparameter tuning.

For these models, we tune the hyperparameters sequentially as a grid search over all the hyperparameters is not possible due to computational limitations. First, a grid search over learning rate and number of neurons in the recurrent layers is done. Using the best values obtained from this, we try different batch sizes to further improve accuracy. Then, we see individually the effect of adding weight sampling and L2 weight regularization on our model. We also test the change in accuracy by removing dropout and learning rate adaptation.

Figure 16: F_1 score variance with number of neurons in recurrent layers.

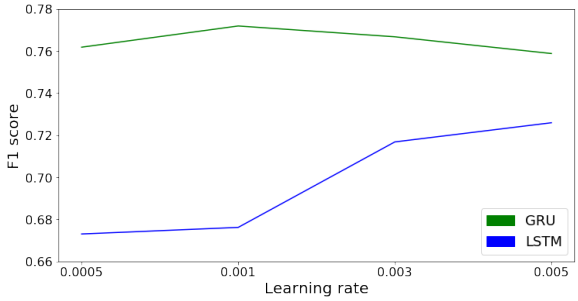


Figure 17: F_1 score variance with learning rate.

Figure 16 shows the variation of the F_1 score with the number of neurons for the best learning rate for both the GRU and LSTM models. Similarly, *figure 17* shows the effect of different learning rates when we fix the best neural structure for the model.
Evaluating the effect of above-mentioned techniques, our final model is with GRU layers containing dropout, adaptive learning rate and L2 weight regularization.

6 Results

We present the performance of the different algorithms described in the previous section. In addition, we look at edge cases to better understand the limitations of the model and do some case studies.

6.1 Final results on different algorithms

Table 6 compares the final results on the test data for our different models.

Model	F1	Accuracy
Random Forest	0.6864	0.7946
SVM	0.5642	0.8273
RNN	0.7714	0.9264

Table 6: Final results (F_1 and accuracy-scores).

For our data set, standard scaling of the continuous features results in better F_1 scores than min-max scaling. We obtain best results using a RNN and see that SVMs and Random Forests are not well suited for our data set. *Figure 18* shows the confusion matrix on the test data for the final result with the best model obtained by the RNN. Here, *lc* denotes lane change sequences and *no lc* are the non lane change sequences. As described in 3.1.3 the test data contains 5000 samples where approximately 17% are labeled as lane change. The precision-score is over 90%. That means lane changes predicted by the model have a very high likelihood to be indeed a lane change. Recall-score is at 62.6%. A reason for that could be the high variance in the features we saw in the statistical data exploration.

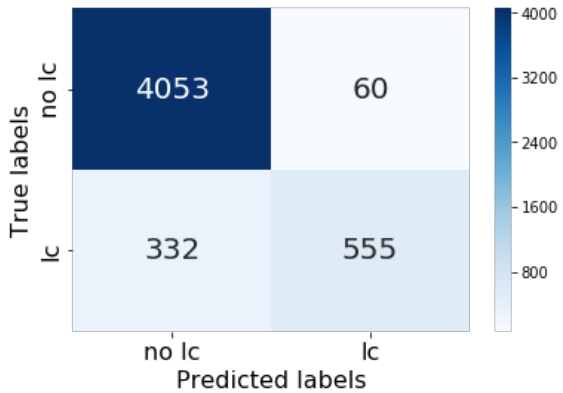


Figure 18: Confusion matrix for the best trained model.

We also fit a RNN by just using the best features obtained from the SVM feature selection. Such a model achieves an F_1 score of 0.7347 with an accuracy of 0.9194. Thus, it works slightly worse than our best model but is a viable option if computational power or memory is restricted.

Additionally, we use the same architecture and hyperparameters of the best RNN to build a new model which also predicts the direction of a lane change. This model has three possible outputs - *lane change to right*, *lane change to left* and *no lane change*. The observed F_1 score is 0.7470 with an accuracy of 0.9014 and the confusion matrix for this is shown in *figure 19*. We conclude that we obtain better results in predicting just the intention of a lane change neglecting the direction.

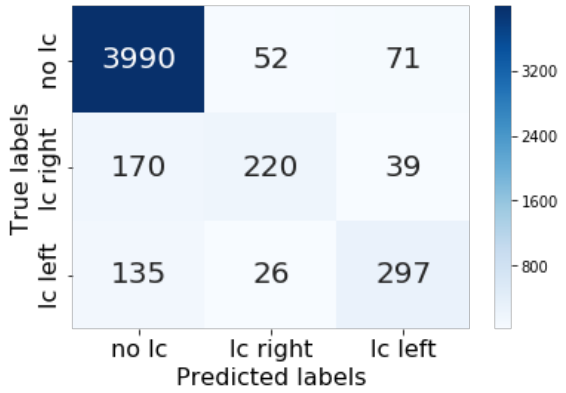


Figure 19: Confusion matrix for the model predicting direction of lane change.

6.2 Edge case exploration

In order to understand the limitations of our model, we consider the *edges* of the confusion matrix in *figure 18*. Therefore, we group the test set by the edges and compute the mean of each feature over the 5 second sequences. Then, we compute correlation factors between these mean values.

As an example, *figure* 20 shows the acceleration of the Ego vehicle where each line represents the mean values of one edge. The mean acceleration increases for the true positives and false positives. The correlation between the mean of these two is 0.74. Therefore, we conclude that for sequences where the acceleration increases the likelihood that the model classifies it as a lane change is higher. That can lead to misclassifications of some non-lane change sequences.

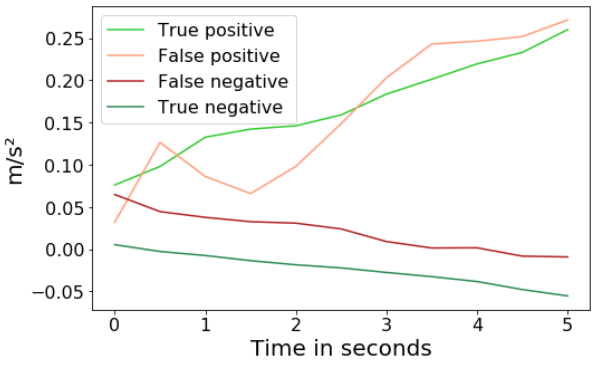


Figure 20: Mean acceleration for all time stamps.

In a similar manner, we study the possible effect on misclassifications for other features. In addition, we take a few edge cases and study the driving videos. We obtain the following observations:

- Driver overtakes but does not go to a free right lane afterwards (false positive).
- Driver changes the lane with no objects around (false negative).
- In construction area, the feature boundary line type is not correctly recorded. Driver cannot go right because the line is solid (false positive).

The list above is clearly not complete as we just look at some samples. Still it can give an idea on the limitations of the model.

6.3 Case studies

In this section, we perform two experiments. First, we apply our best model obtained in *section 5.4* directly to a geographically different market and check its transfer ability. We use exactly the same model and features as before, trained only on German highway data. But now the test set consists of Chinese highway data. The fraction of lane changes vs. non lane changes is identical as in the original test set. We observe that the efficiency on the Chinese data decreases significantly. Therefore, we recommend that a simple transfer of trained models is not practicable. Other than that, the overall method we approach to train the German-data model is transferable.

As a second experiment, we train a model using all the features that we have dropped before including *winker*, *dist_line_r*, *dist_line_l*, *steering_wheel_angle* and *acc_y*. Such a model performs better since these features are highly related to the occurrence of lane changes. But as mentioned earlier, this model no longer predicts the intention of the driver. The results of both experiments are shown in *table 7*.

Experiment	F_1 **score**	**Acc.**	**TP**	**TN**	**FP**	**FN**	**Test samples**
Direct transf. China	0.2664	0.6276	169	1400	700	231	2500
Using all features	0.8828	0.9366	648	4035	78	239	5000

Table 7: Results of experiments.

7 Conclusions and future work

In this project, we developed a data-driven maneuver prediction method for large scaled, real-world traffic data. With the example of lane changes, we invented an algorithm for rule-based detection of these events. Using this algorithm, we automatically generated a ground truth for further implementation steps. Then, we developed a method to extract features from a distributed framework. These features describe the environment and kinematics of the test vehicle (Ego) as well as the objects in the surrounding of the Ego. Next, we implemented three different machine learning approaches using these features and compared their performance. With Recurrent Neural Networks, we obtain an F_1 score of 77.1 % and an accuracy of 92.6 %. We are able to predict lane changes with a high precision. Thus, the algorithm particularly has a very low false positive rate, which means that almost each recognized lane change indeed is a lane change.

One limitation in our data set is the absence of self-representation of the test vehicle, i.e there is no feature which states in which lane the test vehicle is currently driving. So we cannot conclude precisely if there is a free lane in the left or right. Another limitation is that we just have surrounding objects in front of the vehicle, so we do not know the influence of upcoming vehicles from the back. The availability of these attributes should lead to an even higher efficiency of the model.

The final model was trained on German highway data and then directly applied to data from a geographically different area. As we observed a significant decrease of efficiency, we conclude that our model cannot be transferred directly. Instead, we recommend to use transfer learning methods to fine tune our model. Other than that, the overall method we developed is directly applicable to a different market.

In future work, we suggest to include a clustering of the drivers based on similar driving behaviour. This allows developing different models for people with distinct driving patterns. Secondly, using GPS data and navigation systems to better understand the intention behind driving maneuvers should improve the algorithm.

References

[1] Francois Chollet. *Deep Learning with Python*. 1st. Greenwich, CT, USA: Manning Publications Co., 2017. ISBN: 9781617294433.

[2] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[3] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[4] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[5] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. "Visualizing and understanding recurrent networks". In: *arXiv preprint arXiv:1506.02078* (2015).

[6] Gilles Louppe. "Understanding random forests: From theory to practice". In: *arXiv preprint arXiv:1407.7502* (2014).

[7] Hiren M Mandalia and Mandalia Dario D Salvucci. "Using support vector machines for lane-change detection". In: *Proceedings of the human factors and ergonomics society annual meeting*. Vol. 49. 22. SAGE Publications Sage CA: Los Angeles, CA. 2005, pp. 1965–1969.

[8] Tobias Rehder et al. "Effektive Nutzung von hochdimensionalen kontinuierlichen Umfelddaten zur Prädiktion von Fahrverhalten mit Bayesschen Netzen". In: *AAET: Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel ; Beiträge zum gleichnamigen 16. Braunschweiger Symposium vom 12. und 13. Februar 2015, Deutsches Zentrum für Luft- und Raumfahrt e.V. am Forschungsflughafen, Braunschweig*. Braunschweig: ITS Niedersachsen, 2015, pp. 281–301. ISBN: 978-3-937655-34-5.

[9] Dario D Salvucci et al. "Lane-change detection using a computational driver model". In: *Human factors* 49.3 (2007), pp. 532–542.

[10] Julian Schlechtriemen et al. "A lane change detection approach using feature ranking with maximized predictive power". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE. 2014, pp. 108–114.

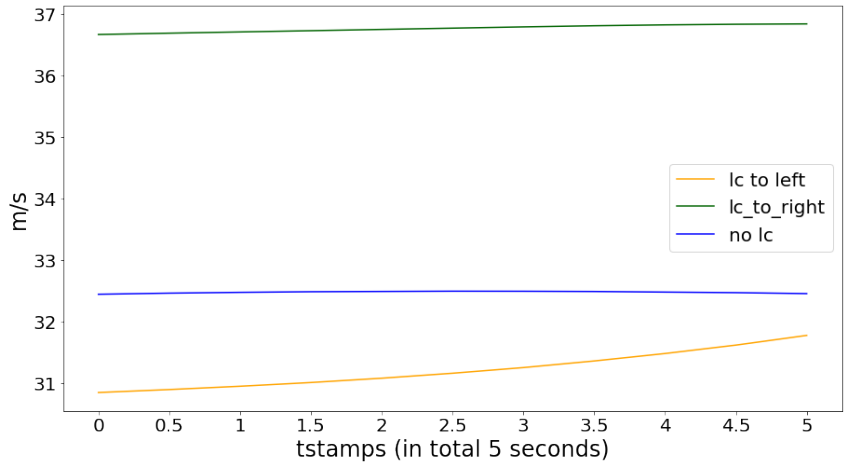
[11] Apeksha Nagesh Shewalkar. "Comparison of RNN, LSTM and GRU on Speech Recognition Data". In: (2018).

Appendix

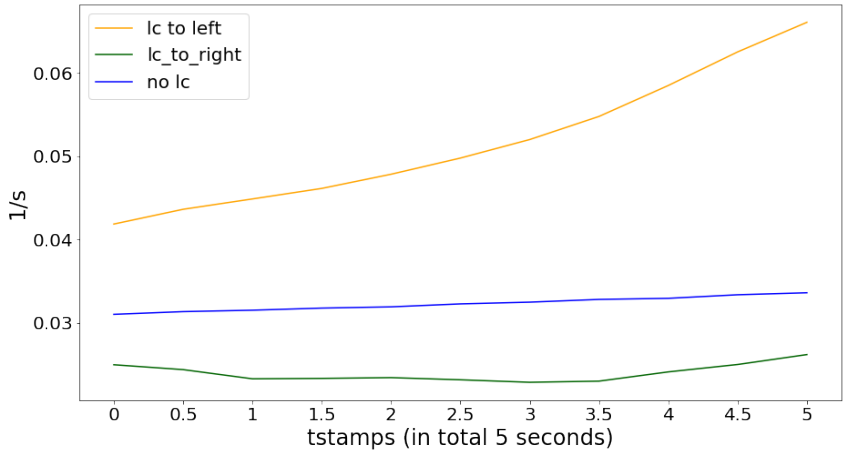
Figure 21: Correlation matrix among all the features.

Feature	**Description**
vel_x	Longitudinal velocity
acc_x	Longitudinal acceleration
road_lanecount	Count of the lanes (in same driving direction)
road_angle	Angle of road
heading	Direction of driving
distance_E1, distance_E2, distance_E3, distance_E4	Distance of different vehicles from the Ego
acc_x_E1, acc_x_E2, acc_x_E3, acc_x_E4	Absolute Longitudinal acceleration of the different object vehicles
vehicles_left_count, vehicles_right_count	Number of vehicles in the left and right lanes
thw_Ego_E1, thw_Ego_E2, thw_Ego_E4, thw_E2_E3	Time headway between two vehicles
inv_ttc_Ego_E1, inv_ttc_Ego_E2, inv_ttc_Ego_E4, inv_ttc_E2_E3	Inverse time to collision between two vehicles
diff_Ego_E1, diff_Ego_E2, diff_Ego_E4, diff_E2_E3	Difference of the speed between two vehicles
E1_exists, E2_exists, E3_exists, E4_exists	Binary feature to show whether a vehicle exists or not
lane_change_r_warning_0, lane_change_r_warning_1, lane_change_r_warning_2, lane_change_l_warning_0, lane_change_l_warning_1 lane_change_l_warning_2	One-hot encoding of the value of the lane_change_warning feature for left and right side of the Ego
boundary_line_l_0, boundary_line_l_1, boundary_line_l_2, boundary_line_l_3, boundary_line_r_0, boundary_line_r_1, boundary_line_r_2, boundary_line_r_3	One-hot encoding of the value of the boundary_line feature for left and right lane

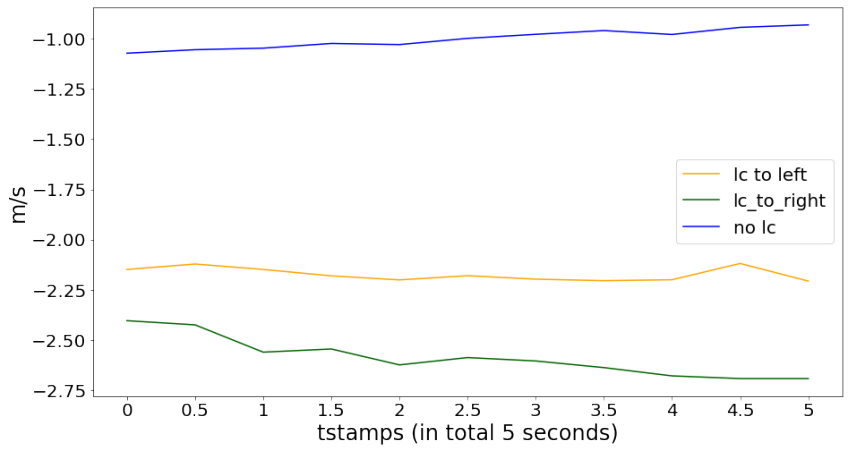
Table 8: Final set of features.



(a) Mean velocity of the Ego vehicle.



(b) Mean inverse time to collision for Ego and E1.



(c) Mean speed difference between Ego and E4.

Figure 22: Data exploration plots for some continuous features.

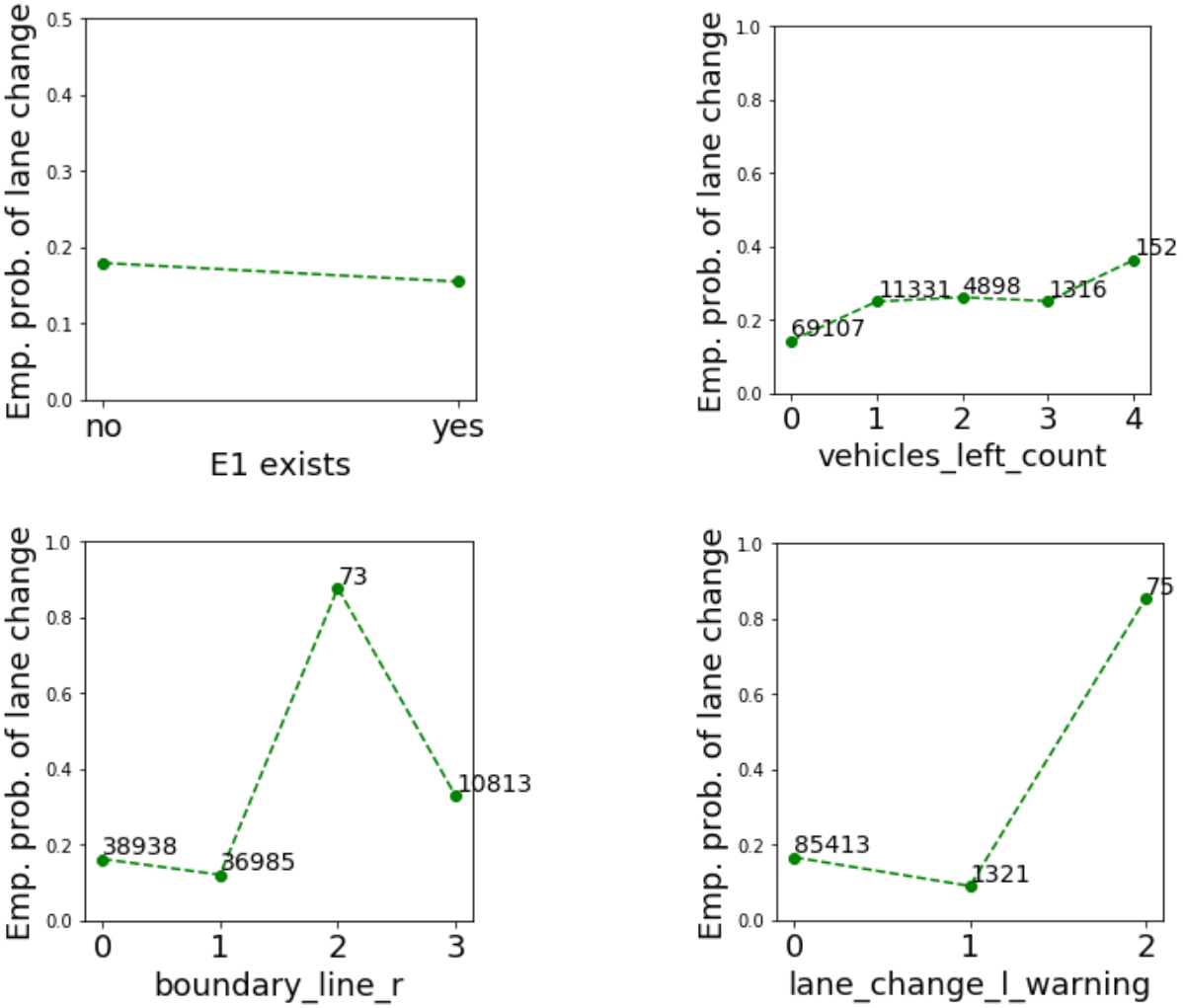


Figure 23: Data exploration plots for some categorical features.