Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Clustering Similar Traffic Scenarios

Murat Can Üste, Zhaoying Chen, Qiaoxi Liu, Emanuel Ramneantu

Practical Course *Motion Planning for Autonomous Vehicles* WS 2019/2020

Advisor:	M.Sc. Moritz Klischat
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Submission:	10. February 2020

Clustering Similar Traffic Scenarios

Murat Can Üste, Zhaoying Chen, Qiaoxi Liu, Emanuel Ramneantu
Technische Universität München
Email: can.ueste@tum.de, qiaoxi.liu@tum.de, zhaoying.chen@tum.de

***Abstract*—The goal of this project is to provide a method to shutil.move(source, dest)find categories of traffic scenarios automatically. The knowledge of traffic scenario categories is important to reduce the validation and testing effort of the development of automated vehicle functions. The architecture consists of three main parts: A data generation part where datasets from different sources are converted to CommonRoad scenario format, a feature extraction part which is applied to the generated CommonRoad scenarios, a clustering technique and a classification technique applied on the extracted features. A modified unsupervised Random Forest algorithm to find a data adaptive similarity measure between all scenarios has been used in this project. The algorithm generates a path proximity matrix, which is a technique to determine similarity based on the Random Forest algorithm. This similarity matrix is then reordered with hierarchical clustering, which results in a graphically interpretable representation. It is shown how the resulting proximity matrix can be visually interpreted and how the variation of the methods' metaparameter reveals different insights into the data. The proposed method is able to cluster data from any data source that can be converted to CommonRoad scenario format. To demonstrate the methods potential, multiple features derived from highway scenarios have been used in this project.**

I. Introduction

The testing and validation of automated vehicle functions take a lot of time and effort. Especially, in the case of kilometer based approach where all the conditions reqired for a given specification has to be mastered. For example, to master the specification given in [1], [2], and [3], billions of kilometers needs to be completed, which is not feasible. In order to overcome this, the scenario based approach is seen as a promising way to test automated vehicle functions [4]. However, there is an infinite number of scenarios that can appear on the road if we look at the scenarios on the microscopic level. Nevertheless, observations from higher level visualizations shows that they follow certain patterns, and can be categorized based on similarity. Therefore, there is a need to know which categories there are for the scenarios that appear on the road, which will reduce the testing and validation effort while ensuring broad scope by testing representatives from each category.

The main purpose of this project is to implement a process of categorizing/clustering of traffic scenarios in an unsupervised way. The implementation presented in this work are separated into three main parts as it can be seen on Figure 1. The first part is the data generation where datasets from different sources are converted to CommonRoad scenario format [5], which provides a common format for the scenarios, as well as a benchmark for the motion planning algorithms.

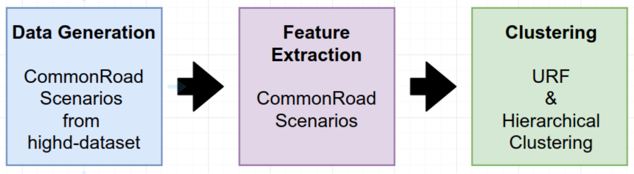

Data Generation
CommonRoad Scenarios from highd-dataset
Feature Extraction
CommonRoad Scenarios
Clustering
URF & Hierarchical Clustering

Fig. 1: The overall architecture.

The second part is the feature extraction part where the generated CommonRoad scenarios are processed in order to create a feature model. The scenario quantitative model is inspired by the idea from [6]. For this report, 5 specific time steps are chosen and the corresponding state of ego and surrounding vehicles are calculated.Totally 57 features are defined and implemented to describe the scenarios quantitatively. Furthermore, the importance of features is analysed, which will help feature selection in the clustering part.

Clustering is an unsupervised data exploration task. This is in contrast with supervised learning for which the datasets are labelded with classes. We study clustering in a high dimensional numerical and categorial mixed space. The traditional methods for clustering (like kmeans clustering) have some shortcomings. In our report, we are inspired by the idea from [7] using so called modified Unsupervised Random Forest to implement our algorithms as well as a handy interface to interact with raw dataset. The algorithms mainly are composed of two core techniques: the clustering-based decision tree and proximity matrix (proposed in section IV). According to the test results from the first version of implementation, we notice some drawbacks and based on that we improve the details which are introduced in section V.

In summary, the work presents a method which enables the clustering/categorization of scenarios from different data sources, and provides a way to investigate relations between them.

The paper is organized as follows: The data generation process, as well as the data source used are described in section II. Then, the feature extraction process and the description of the extracted features are given in section III. Afterwards, the Unsupervised Random Forest, and Hierarchical Clustering methods are explained in section IV. In section VII, some results of the method based on the generated datasets are presented. Finally, this work is concluded in section VIII.

II. Data generation

In this section, we explain the dataset we used for this project and the data generation process in detail.

There are several datasets with the intented purpose of traffic simulation such as NGSIM [8] [9], or automated driving research such as KITTI [10] and Cityscapes [11]. The KITTI and Cityscapes datasets are mainly used for the development of computer vision algorithms since they contain annotated images from vehicle onboard cameras, whereas the NGSIM dataset focuses on vehicle trajectories on highways and urban traffic roads captured from bird's eye view. However, we use the highD dataset [12] in this project due to the fact that even though it is similar to the NGSIM dataset as both are captured from bird's eye view and focus on highway scenarios, the highD dataset has the advantage of capturing the traffic from high longitudinal and lateral accuracy due to the usage of aerial drones, which results in lower positioning error.

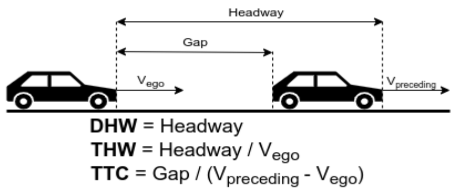
The highD dataset already provides pre-extracted information about the vehicles, trajectories and important metrics. Even though these information can be used for clustering scenarios as is, we convert the highD recordings into CommonRoad scenario format [5] in order to have a common format for the scenarios, and divide the recordings into small segments in order to capture traffic scenarios in the microscopic level. Another advantage of this approach would be the possibility to extend and improve the feature extraction and clustering steps later on with the usage of scenarios from wide variety of sources converted to the CommonRoad format. This would be specifically inline with the purpose of clustering and categorizing traffic scenarios in order to reduce testing and validation effort.

In the following subsections, we explain and analyze the highD dataset, and move on to the explanation of filtering and CommonRoad scenario generation processes. Lastly, we analyze the generated datasets before moving on the the feature extraction section.

A. *The highD Dataset*

The highD dataset is a dataset of naturalistic vehicle trajectories recorded on German highways. The recordings were done using an aerial drone equipped with a high resolution camera. Usage of aerial drones has the advantage of capturing the traffic from a bird's eye view with high longitudinal and lateral accuracy. Even though the information about the object heights are lost, it doesn't effect the safety validation since object heights have limited relevance. In our case, this has no effect on the resulting scenarios, since CommonRoad Scenario format also uses bird's eye view.

The recordings were done at six different highway locations in Germany around Cologne. There are total of 60 recordings, with each recording being average length of 17 minutes, which include more than 110.500 vehicles, and 147 driven hours. The road segments are about 420m length, and the number of lanes in the recordings are either 4 or 6 depending on the location, with half of them going in reverse direction compared to the other half. The dataset also contains pre-extracted information such as vehicle types, vehicle trajectories, metrics such as DHW, THW, and TTC, as well as maneuver information such as lane changes [12].



Headway
Gap
V_{ego}
$V_{preceding}$
DHW = Headway
THW = Headway / V_{ego}
TTC = Gap / ($V_{preceding}$ - V_{ego})

Fig. 2: Calculation of DHW, THW, and TTC.

The DHW, THW, and TTC metrics have great importance when deciding on a scenario's criticality, and we will make use of them heavily throughout this paper. The calculation of DHW, THW, and TTC metrics are shown in Figure 2. The definitions of them as follows:

- *Distance Headway (DHW):* The distance between the front of the ego vehicle and the front of the preceding vehicle in the same lane.
- *Time Headway (THW):* The time required for the ego vehicle's front to reach the same position as the preceding vehicle's front, if it continues with its current speed.
- *Time to Collision (TTC):* The time required for there to be collision between the ego vehicle and the preceding vehicle, if they continue with their current speed and follow the same path.

In Figure 3, we can see an example segment from the recordings which was visualized using the tool provided at [13]. The red rectangles indicate the vehicles on the road, and the red lines indicate the trajectories followed by vehicles. We can also observe the vehicle type, speed of the vehicle, and the ID of the vehicle in the yellow labels.

According to [12], there are 4 main maneuvers detected on the dataset by following the rules in [14] for the critical maneuvers:

- *Free Driving (Longitudinal uninfluenced driving):* Driving without being influenced by a preceding vehicle
- *Vehicle Following (Longitudinal influenced driving):* Actively following another vehicle
- *Critical Maneuver:* Low Time to Collision (TTC) or Time Headway (THW) to a preceding vehicle
- *Lane Change:* Crossing lane markings and staying on a new lane

These maneuvers were detected using predefined set of rules and thresholds. We will make use of these maneuvers during our filtering process in order to discard scenarios that pose no significant challenges to the motion planning algorithms.

B. *Converting highD dataset to CommonRoad Scenarios*

In order to convert highD recordings to CommonRoad Scenarios, we inpect each vehicle with their trajectories and consider the most critical maneuver performed by the vehicle as a scenario candidate during the conversion process. The

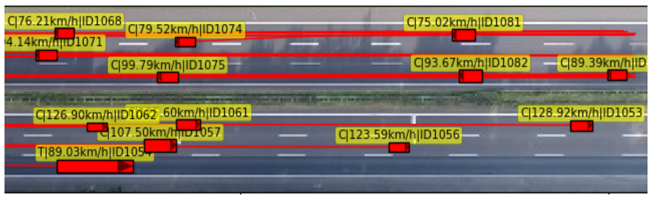


Fig. 3: An example segment from the highD recordings using the visualization tool provided at [13].

order of criticality is as follows; the *Lane Change* maneuver is more critical then *Critical Maneuver*, followed by *Vehicle Following* and then *Free Driving*.

In Figure 4, we can see the percentages of the mentioned maneuvers performed by vehicles after assigning each vehicle to its most critical maneuver group. The *Lane Change* maneuvers are filtered based on the number of lane changes defined for the vehicles in the dataset, which means we select the vehicles that performed at least one lane change during the recording. The vehicles with no *minimum DHW* defined were assigned to the group of *Free Driving* maneuvers. The vehicles that didn't perform any lane changes, and had *minimum DHW* defined were assigned to the *Vehicle Following* and *Critical Maneuver* groups, where the latter group having *minimum THW* lower than 1.75 seconds and/or *minimum TTC* lower than 0.5 seconds. These thresholds were taken from [14], as the highest thresholds for a scenario to be identified as a *critical maneuver*.

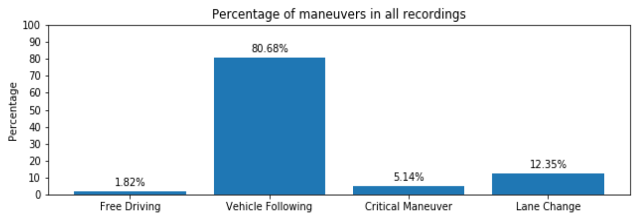


Fig. 4: Percentage of detected maneuvers on the highD dataset.

One thing to note here is that, the DHW, THW, and TTC metrics were calculated without a distance threshold in the dataset. If we apply a distance threshold in the dataset, for example a sensor range, the percentages of the maneuvers on Figure 4 would change drastically, moving significant portion of *Vehicle Following* group over to the *Free Driving* group.

Since the *Free Driving* scenarios pose no significant challenge to the motion planning algorithms, we try to filter out as much of them as possible, and increase the percentage of other critical maneuvers. In order to do so, we apply a filtering process on the data before we move on to the actual scenario generation.

1) Filtering Process: As mentioned before, we inspect each vehicle and their trajectories in the dataset and consider their most critical maneuver as a candidate for a scenario. This is done by the usage of metrics DHW, THW, and TTC, as well as the lane change information of the vehicles. Following steps are performed on each recording to filter out critical scenarios based on lane change information and one of the DHW, THW, or TTC metrics of each vehicle:

- *Step 1:* Filter out trajectories that don't have the selected metric defined.
- *Step 2:* Filter out trajectories that don't perform lane change maneuver if we want to use the lane changing vehicles only.
- *Step 3:* Sort each trajectories' frames based on selected metric, and select the time step with minimum value for each trajectory.
- *Step 4:* Generate the scenarios by setting the selected time steps as the middle time step of a scenario.

We decided to have total of 81 time steps in our generated scenarios, which will result in 3.24 seconds long segments. Since the recordings have 25 frames per second, the $\Delta time$ between each time step is 0.04 seconds. The timespan of 3.24 seconds, or 81 time steps, was decided after visualizing the extracted scenarios, and observing that in most of the scenarios, the maneuvers performed by the ego vehicle (i.e lane change) was finalized within this time span.

To give an example of a filtering process, we can select DHW metric for criticality, and use only the lane changing tracks. After we filter out the tracks that don't have the DHW metric defined, and don't perform lane changes during the recording, we would be left with a subset of vehicles from the recording. Afterwards, we would sort each vehicles' trajectory states based on dhw metric, and select the time step with the lowest DHW value for each vehicle. Each of these vehicles, and their corresponding time steps, would be the ego vehicle of a scenario, and the middle time step of the scenario respectively, if they are not discarded during the scenario generation process.

After selecting the most critical time step for each vehicle in the recordings after filtering, we move on to the CommonRoad Scenario generation process.

2) CommonRoad Scenario Generation Process: After the filtering process is done, we are left with sets of vehicles and their most critical time steps for each recording. Setting the associated time steps as the middle time steps of the scenario candidates, we calculate the initial and final time steps of the scenario by going 40 frames back, and forward in the recording. Afterwards, we inspect the segment between these frames of the recordings to discard scenario candidates that are not fit to be converted to CommonRoad scenario format. The main steps of scenario generation as follows:

- *Step 1:* Discard the scenario if there are not enough time steps for the selected trajectory when the associated frame is selected as the middle time step.
- *Step 2:* Discard the scenario if the initial or final position of the ego vehicle is too close to the scenario's edges,

in order to have meaningful features during the feature extraction process.

- *Step 3:* Discard the lanelets and trajectories that are in the opposite direction. Since the lanelets in opposite directions are physically seperated, they have no effect on the scenario, and would provide unnecessary information.
- *Step 4:* Generate dynamic obstacles from all vehicles within the selected initial and final time steps.
- *Step 5:* Set the obstacle ID of the dynamic obstacle that corresponds to the ego vehicle to a unique number in order to differantiate it from rest of the obstacles (99 was assigned in our case).

As it can be seen, we look whether the selected vehicle has enough time steps in its trajectory for the defined segment, and whether its initial and final time step positions are too close to the scenario's edges. The positional check is done in order to make sure we can extract meaningful features during the feature extraction process. This will be explained in detail on section III.

Additionally, we remove the dynamic obstacles within the defined segment if they don't have at least 10 time steps defined in their trajectories. This means that, the vehicles go out of the scenario during the inital time steps, or come into the scenario during the final timesteps are dicarded, since they have no effect on the scenario and have very little information.

In Figure 5, we can see two examples of a generated CommonRoad scenarios. Both of these scenarios were taken from the dataset that was generated using TTC as metric, and only lane changing vehicles. The figures show each scenario's most critical time step (41st time step) based on the ego vehicles. In Figure 5a, the vehicle performs the *Vehicle Following* maneuver throughout the scenario, and the figure shows the *minimum TTC* time step for the ego vehicle. In Figure 5b, the vehicle performs the *Lane Change* maneuver, and the figure shows the *minimum TTC* time step for the ego vehicle. In both cases, the lanes and vehicle going in reverse direction were removed from the scenario.

C. Generated CommonRoad Scenario Datasets

After applying the filtering and scenario generation processes to the highD dataset with three different metrics and two different cases of including only lane changing vehicles, and including all vehicles on the dataset, we generated 6 different CommonRoad scenario datasets. Including only the lane changing tracks during generation of scenarios results in smaller datasets since we are only considering 12.35% of the vehicles in the dataset as it can be seen on the Figure 4. In the case of including all vehicles, we end up using 98.18% of the vehicles on the dataset. Since we are filtering out the vehicles where the selected metric was not defined, the *Free Driving* scenarios are being filtered out as they are not following a preceding vehicle, meaning that none of the metrics are defined for them.

We can see the combination of parameters used for filtering, and the resulting dataset sizes in Table I. As the datasets generated using only the lane changing tracks result in having a smaller set of generated dataset, we named them by appending 'Small' after the selected metric's name. And similarly we named the datasets generated using all vehicles by appending 'Big' after the selected metric's name.

TABLE I: Generated CommonRoad Scenario Datasets

Dataset Name	Metric	Only Lane Changing	Sample Size
DHW Small	Min DHW	True	6.551
THW Small	Min THW	True	6.544
TTC Small	Min TTC	True	6.944
DHW Big	Min DHW	False	44.812
THW Big	Min THW	False	44.107
TTC Big	Min TTC	False	44.280

The number of scenarios generated for the small datasets are 6.680 on average, which means that we are discarding at least half of the scenarios during the filtering and scenario generation processes, as the number of vehicles that performs *Lane Change* maneuver should be around 13.647 (there are 110.500 vehicles in the dataset, 12.35% of them are performing *Lane Change* maneuver).

The number of scenarios generated for the big datasets are 44.400 on average, which means that we are discarding more than half of the scenarios during the filtering and scenario generation processes, as the number of vehicles should be around 108.489 excluding the *Free Driving* maneuver vehicles (there are 110.500 vehicles in the dataset, and we consider 98.18% of them).

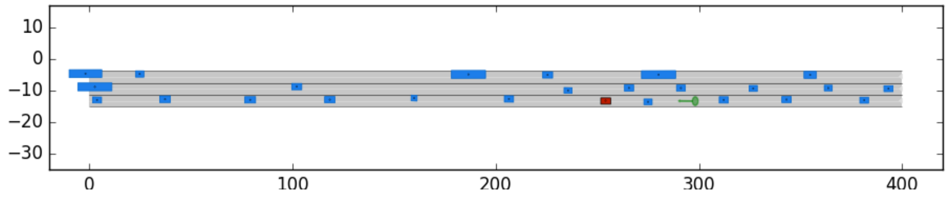
In order to inspect the generated datasets in detail, we analysed the maneuvers in each dataset in a similar manner to the Figure 4 using the same set of rules and thresholds after the feature extraction was performed on all of them.

In Figure 6, we can see the percentages of detected maneuvers on each small datasets. Even though the vehicles that performed *Lane Change* maneuver were selected for the generation of the small datasets, the resulting maneuvers are not all *Lane Change* maneuvers. This means that even if the vehicle performs a lane change, its most critical time step in terms of the selected metric might not necessarily be during the lane change maneuver.

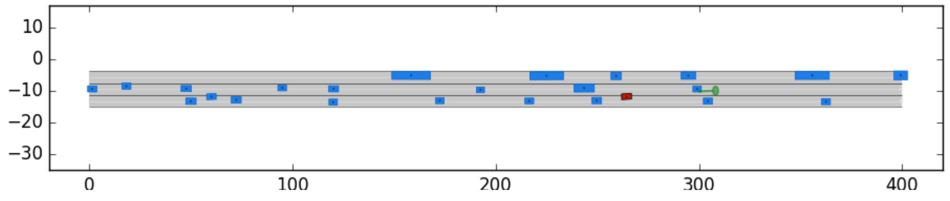
Other important difference in the ratio of the maneuvers is that the ratio of *Free Driving* maneuver is a lot higher compared to before. This is the result of applying a distance threshold in order to simulate sensor range during the feature extraction process, meaning that the *Vehicle Following* and *Lane Change* maneuvers where the preceding vehicle is outside of the sensor range (100 meters in our case) are considered as *Free Driving* from ego vehicle's perspective.

Similarly, in Figure 7, we can see the percentages of detected maneuvers on each big datasets. In this case the ratio of each maneuver is similar to the original dataset, except the *Free Driving* maneuvers, which was caused by the distance threshold explained above, which results in having lower ratio for *Vehicle Following* and *Lane Change*.

Another thing to notice here is that the ratio of *Critical Maneuver* is much lower in all datasets compared to the



(a) An example of *Vehicle Following* maneuver on the generated datasets.



(b) An example of *Lane Change* maneuver on the generated datasets.

Fig. 5: Visualization of two examples from the generated dataset with TTC as the metric, and considering only the lane changing vehicles. The figures show each scenario's most critical time step based on the ego vehicle. In (a) we can see an example of *Vehicle Following* maneuver, whereas in (b) we can see an example of *Lane Change* maneuver performed by the ego vehicle.

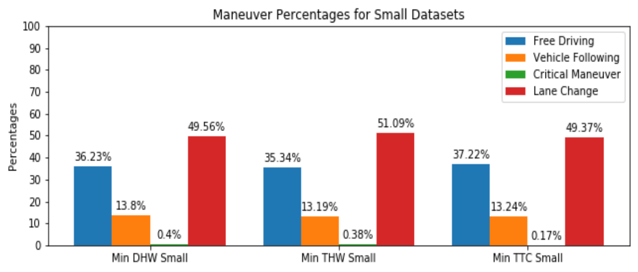


Fig. 6: Percentage of the detected maneuvers on the generated datasets with considering only lane changing vehicles.

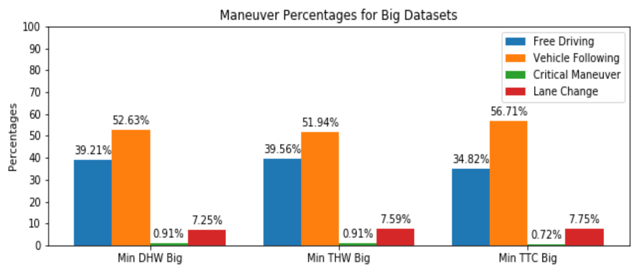


Fig. 7: Percentage of the detected maneuvers on the generated datasets with considering all vehicles except the ones with *Free Driving* maneuver.

original. This is due to the scenario generation process where we discard scenarios where there are not enough time steps in the ego vehicle's trajectory, and the position of the inital or final state of the ego vehicle being too close to the edges.

III. Feature extraction

The main purpose of feature extraction is to extract typical characteristics of the scenarios and build a quantitative model for them. Based on the actual highway model, the area around the ego is divided into six zones and we apply a sensor range of the ego vehicle (100 meters in our case). It is assumed that only the vehicles inside the sensor range of ego vehicle would influence the maneuver of ego vehicle and only the nearest surrounding vehicles on or next to the lane where ego vehicle is would directly influence the ego vehicle. The proposition is depicted in Figure 8.

A. Feature Model

As mentioned before, the minimal THW, DHW and TTC are the metrics to determine the criticality of scenarios, because the states of vehicles represent more typical characteristic of scenarios at these timesteps. Therefore, the features are set up on five timesteps: At the beginning, the minimal THW, the minimal DHW, the minimal TTC and the end. To describe the traffic density, we consider the relative distance between ego vehicle and its surrounding vehicles. Also, we calculate how many surrounding vehicles there are.

Other features are defined according to the state of ego vehicle and we observe its changes over time. On one hand, we keep record some basic states of ego vehicle including acceleration and absolute velocity. On the other hand, we consider its three typical maneuvers, including lane change, cut-in and brake to describe the mainly moving pattern of the ego vehicle and further describe the characteristics of the scenarios.

In total a set of 57 features are defined and calculated to be fed in the clustering process. The feature structure is shown in Figure 9. The description of all features is in the appendix.

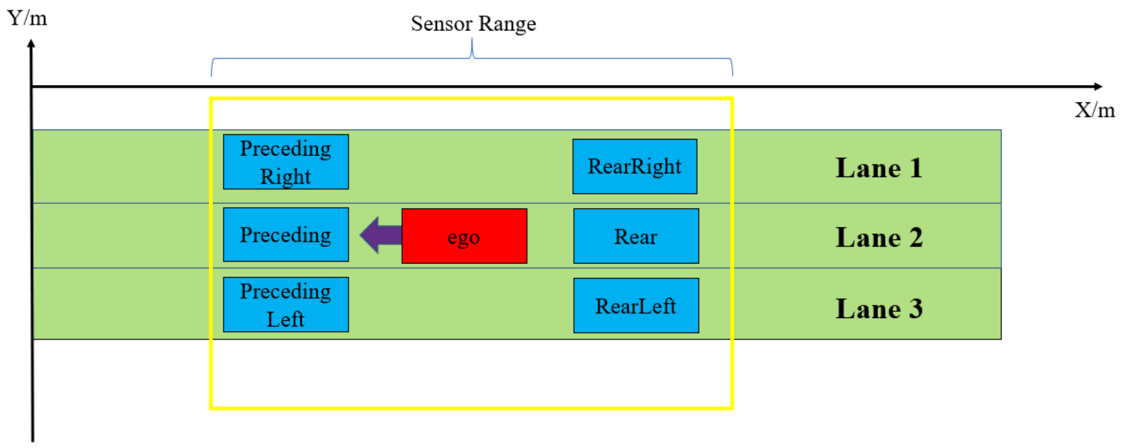
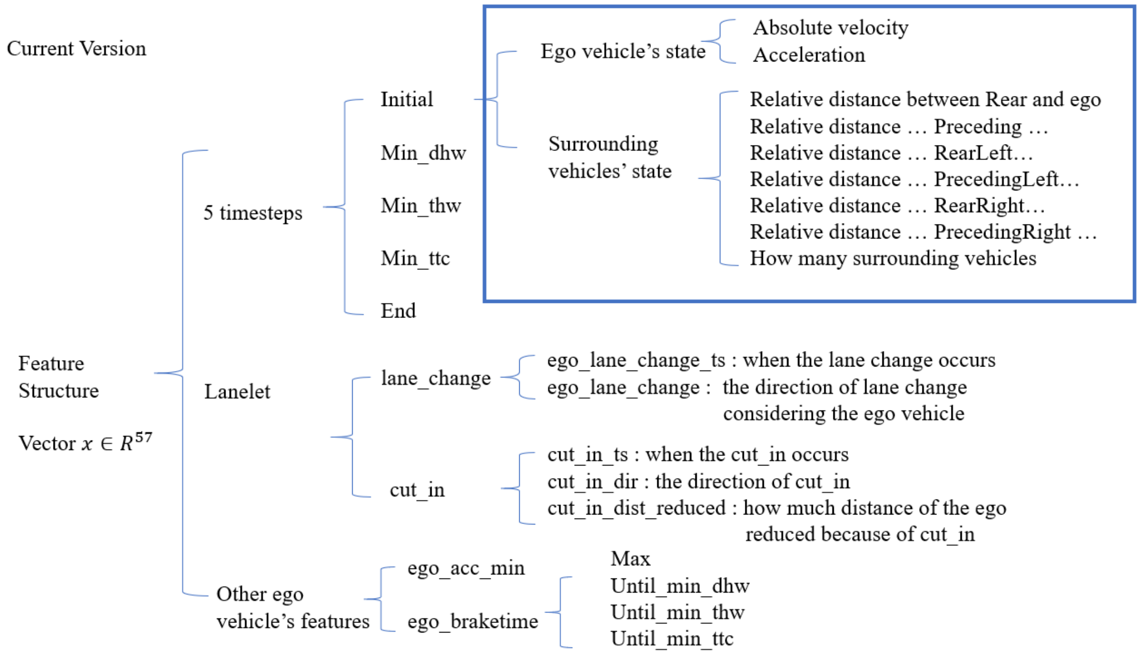


Fig. 8: Only the surrounding vehicles (preceding right, preceding, preceding left, rear right, rear and rear left vehicles) in the sensor range (yellow box) are considered in the scenario model. In our paper, the size of sensor range is 100 meters.



Current Version
Feature Structure
Vector $x \in R^{57}$
5 timesteps
Initial
Min_dhw
Min_thw
Min_ttc
End
Ego vehicle's state
Absolute velocity
Acceleration
Surrounding vehicles' state
Relative distance between Rear and ego
Relative distance … Preceding …
Relative distance … RearLeft…
Relative distance … PrecedingLeft…
Relative distance … RearRight…
Relative distance … PrecedingRight …
How many surrounding vehicles
Lanelet
lane_change
ego_lane_change_ts : when the lane change occurs
ego_lane_change : the direction of lane change considering the ego vehicle
cut_in
cut_in_ts : when the cut_in occurs
cut_in_dir : the direction of cut_in
cut_in_dist_reduced : how much distance of the ego reduced because of cut_in
Other ego vehicle's features
ego_acc_min
ego_braketime
Max
Until_min_dhw
Until_min_thw
Until_min_ttc

Fig. 9: The feature structure.

B. Analysis of Crucial Features

A feature vector can include all the 57 features to describe a scenario. However, it will be time-consuming when clustering because some relatively less important features are considered. To strike a balance between describing the scenarios accurately and decreasing the computation when clustering, we need to analysis the crucial level of certain kinds of features. After we know which features are more crucial, we can only choose those features to form a feature vector used in the clustering part.

Based on the Random Forest algorithm, when splitting trees, we choose the features with highest *Gini Gain* as the splitting feature. The more frequent a feature is chosen, the more important it will be. We take an experiment in the dataset with *dhw-only-lanchanging* metric, building 300 trees with maximal depth of tree = 5. The Figure 10 shows the statistics about how many times that each feature to be chosen.

We separate the relative features into various groups and calculate the average chosen times of each group of features. 7 groups are defined as shown in figure 11.

The order of crucial features is as follows:

- Compared the state of ego vehicle:
  ego lane change >brake time>acceleration>cut in
- Compared the state of ego vehicle and its surrounding vehicles:

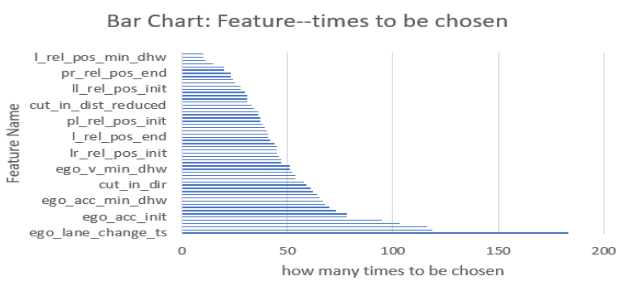


Fig. 10: There are totally 2814 splits. The most frequently chosen feature is *ego-lane-change-ts* (when the ego lane change occurs) with 183-time chosen and the less frequently chosen feature is the relative position between rear vehicle and ego vehicle at minimal dhw, with 10-time chosen.

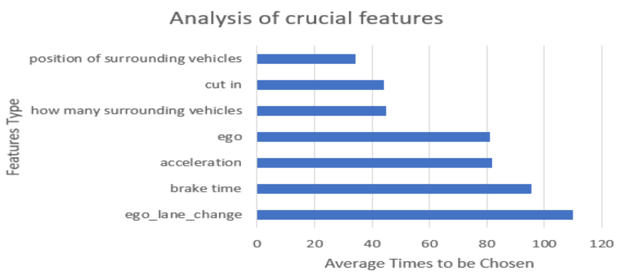


Fig. 11: The average chosen times of each grouping features.

ego vehicleâs state>surrounding vehiclesâ state

Thus, if we need to decrease the size of feature vector for faster computation, we can ignore certain features based on the statistics. Specially, the relative position of the surrounding vehicles is less crucial.

IV. Unsupervized Random Forest

The aim of our task is to cluster similar traffic scenarios in an unlabeled dataset. As mentioned before, we defined more than 50 features on the data set which is high dimentional. Furthermore, the dataset also have different types of data, namely, numerial and categorial. Besides, the data also varies according to the feature. Therefore, we apply the idea from random forest (RF) to group the relevant scenarios. Since the classical application of RF is introuduced for supervised learning, it leads to a similarity measurement between labeled classes. [15] shows that RF predictor also works on unlabeled data well by generating synthetic data. Here we describe the properties of unsupervised RF method and how we implement for clustering traffic sccenarios in practice.

A. Motivation of Random Forest(RF)

RF is a well known ensemble predictor of individual classification tree predictors. The URF method distinguishing the original data from the synthetic data yields a similarity measure that can be used as input in further unsupervised learning algorithms. The basic principle is, each tree votes for a classification and the forest chooses the classification which has the most votes over all the trees. Compared with tranditional RF, the difference of URF is constructing each binary decision tree by adding an argumented dataset as labeled datapoints. Advantages of URF:

- It can handle mixed variables (numerical and categorial).
- It gives estimate of which features are important. (explainable for feature engineering)
- As ensemble learning method, it reduces overfitting and maintains robust for missing data and outliers.
- It computes proximities between pairs of cases that can be used in clustering.

B. Challenges of classical RF

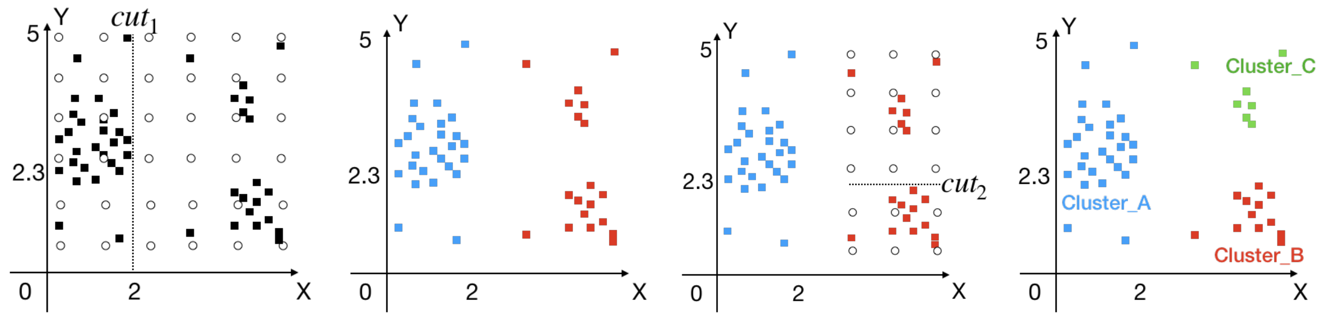
1) Unlabeled dataset: However,the main problem of URF is: If the original data distributed in high dimensional spaces is not dense or well balanced, based on this, Generating synthetic data leads to an exponential number of synthetic datapoints.[7] In order to overcome the drawback of URF, a method called CLustering Tree (CLTree) is introduced in [16]
Instead of adding augmented dataset in fully dimensional space, the CLTree technique creates uniformly distributed points in each split where only one feature (one dimension) determines the splitting result. Inspired by CLTree, we impemented our Modified URF algorithm in Python for our clustering scenarios task based on constructing decision tree for the binary classification.

2) Similarity Measurement: After detecting a URF clustering result, we have to consider how (dis)similar among them. For labeled dataset, the terminal leaves in which the datapoints are landed are considered. If two obeservations placed in the same leave node, during accumulating the result of forest, the similarity between them is increased by one. Finally, the average similaries between each pair of datapoints is calculated and can be applied for further distance measurement. In [6] a proximity measure was introduced. Intuitively, each scenario starting from the root node, went through some decision node and at the end arrived a leave node. The whole journey of one datapoint reflect the property of this datapoint since each decision node has its criterion feature and corresponding value. Thus, the difference between two datapoints is the different path they traveled. The more overlapped nodes they passed by, the more common properties they have, the more similar they are. Concrete solution is illustrated later in subsubsection IV-C3

C. Solution and Implementation

This part presents how we apply modified URF algorithm by generating synthetic data as well as similarity measurement method in practice to achieve our goal.

1) modified Decision Tree (DT) Construction: As mentioned in subsubsection IV-B1, in order to transfer our unsupervised task (clustering) into a supervised method, our goal turns to separate synthetic from original data. Since it is principally a decision tree which perform a binary split, we modify the algorithm and use Gini purity as splitting criterion.



(a) first cut (b) clean synthetic data (c) second cut (d) final cluster result

Fig. 12: Simulation of generating synthetic data and splitting on two dimensional data. (a) shows the first split by feature X at the value 2. Because on the left of cut_1, it has a majority of black points whereas on the right side it has more white points. Finished first split, white points are deleted. the process for second split by feature Y is similar as the first. After two times split, all the datapoints are loacted in one cluster.

In python we define a Class $DecisionNode$ which is the component of a tree. It has several attributes: the splitting feature, the splitting value, the result which could be another $DecisionNode$ or a dictionary composed of clustered points.

The input for a DT in our task is a set of artificially labeled datapoints which consists of two part: scenearios from original dataset (labeled as 0) and synthetic points uniformly generated.

A decision tree has two types of nodes, decison nodes ($DesicionNode$) and leaf nodes (the dictionary stored in its $DesicionNode$'s result). A decison node which can also regarded as subtree includes the decision feature and two branches. Thus we can build a tree recursively: (See Algorithm 1)

Algorithm 1 BuildTree($df, mLeaves, gini, minMax, col$)

```
1:  if $length(df) == 0$ then
2:      $DecisionNode()$
3:      if $length(df) > mLeaves$ then
4:          $createfakeData(df, minMax, col)$
5:          $selectLocalBestGini$
6:          $selectGlobalBestGini$
7:          $bFea, sValue, bGini <- selectGlobalBestGini$
8:          if $bGini >= gini$ then
9:              $set1, set2 <- split(df, bFea, sValue)$
10:             $branch1 <- BuildTree(set1, ..)$
11:             $branch2 <- BuildTree(set2, ..)$
12:             return $DecisionNode()$
13:         end if
14:         return $DecisionNode()$
15:     end if
16: else
17:     return $DecisionNode()$
18: end if
```

1) Initial state: N obeservations with M features, number of datapoints becoming a leaf (for pruning) is L.
2) Sample $r * N$ obeservations randomly (ratio r is a hyperparameter) with replacement and select randomly $m = \lfloor \sqrt{M} \rfloor$ number of features $f_1, f_2, ..f_m$.
3) If the number of datapoints is smaller than L, Stop growing.
4) Generate synthetic data (uniformly distributed) randomly for each selected feature dimension by the minimum and maximum values over original data ranges.[7]
5) Calculate best split value bs_i using Gini index as splitting criterion for each chosen feature f_i.
6) Compare with all best split values of features and choose $bValue = min(bs_1, ..., bs_m)$ as the global best split value as well as corresponding feature f_{split}
7) Deleted all synthetic data and divide dataset by f_{split} and $bValue$ into left and right branch.
8) loop back to the third step.

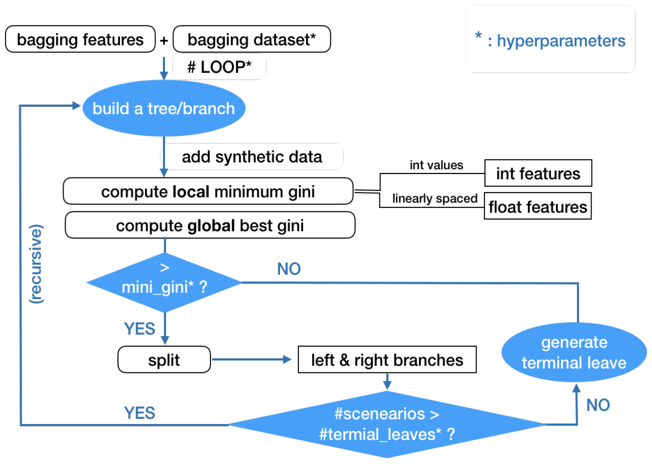


Fig. 13: Diagram of Modifed URF Algorithm

2) Preorder Tree Traversal: After constructing the most important conponement,the individual DT, we start to implement

the complete URF algorithm.
The root node ($DecisionNode$) of each tree in the forest contains a bootstrap sample from the original data as the training set. The leave nodes belonging to their parent $DesicionNode$ store the clustered datapoints. That is to say, in order to apply the result of a DT for statistic, we write a recursive function $PreTraversal$ which is responsible for pre-order tree traversal. The algorithm which has only three basic steps is consise combined with our self-defined class $DesicionNode$. (See Algorithm 2) The input is a $DecisionNode$ which can either a tree or a subtree while output is a dictionary in which the key is the unique datapoint ID and the corresponding path (defined in subsubsection IV-C3) is the value. Main steps on $PreTraversal$ are as following:

1) visit the node, check wether it is a leave node. If not, then continue; Otherwise return the stored datapoints, End.
2) visit its left branch (recursively)
3) visit its right branch (recursively)

Algorithm 2 PreTraversal($tree, pos, res, depth = 0$)

```
1: if $tree.value! = None$ then
2:     $pos_l = pos +' L'$
3:     $pos_r = pos +' R'$
4:     if $tree.left.res! = None$ then
5:         $res[pos_l] = tree.left.res$
6:     end if
7:     if $tree.right.res! = None$ then
8:         $res[pos_r] = tree.right.res$
9:     end if
10:    $PreTraversal(tree.left, depth + 1, pos = pos_l, res)$
11:    $PreTraversal(tree.right, depth+1, pos = pos_r, res)$
12: else
13:    $res[pos] = tree.res$
14: end if
15: return $res$
```

3) Implementation of Proximity Matrix : Since we yield a forest according to modified URF, datapoints(scenarios) are landed in multiple trees.
In the forest, a tree T^k is built based on our dataset D. We define a path of a datapoint (scenario) x in T^i through all decision nodes as

$$p_x^k = \{n^0, n_{pos}^1, ..., n_{pos}^m, \}$$

where pos is either left or right, m represents the depth of this tree.
In our work, we apply Jaccard Index for calculating the similarity score between pair of datapoints x and y. Namely,

$$score_{x,y}(T^k) = \frac{|p_x^k \cap p_y^k|}{|p_x^k \cup p_y^k|} \in (0, 1]$$

Because of ensemble predictor, at the end we need average all trees in the forest. Assume that the pair of x and y appears N times in N trees, the final proximity score of it formulated as:

$$Score_{x,y} = \frac{1}{N} \Sigma_{k=1}^{k} score_{x,y}(T^k)$$

Algorithm 3 demonstrates how we implement. The variable $forest$ represents the result which is a list consisting of all the return value from Algorithm 2. $forest$ is the total number of datapoints. m, M stands for the matrices storing (sum of) scores for individual tree and forest respectively while matricies C, c record the frequencies for each pair. The output is also an triangular matrix whose entry (i, j) depicts $Score_{i,j}$ in terms of symmetry. Based on this output, we can compute dissimilarity (distance) matrix as requested for clustering.

Algorithm 3 BuildPMatrix($forest, size$)

```
1: $M < -initZeroMatrix(size)$
2: $C < -initZeroMatrix(size)$
3: for $tree \in forest$ do
4:     $m < -initZeroMatrix(size)$
5:     $c < -initZeroMatrix(size)$
6:     for $i, x \in tree$ do
7:         for $y \in tree[i :]$ do
8:             $m[x.index][y.index] < -computeScore(x, y)$
9:             $c[x.index][y.index]+ = 1$
10:        end for
11:    end for
12:    $M+ = m$
13:    $C+ = c$
14: end for
15: return $M/C$
```

V. Problems and Improvements

In the following we will present some of the shortcomings of the initial URF algorithm and our attempts to fix those, by updating specific parts of the implementation. We also give an overview of the performance of the two versions on a limited dataset.

A. Shortcomings

As we tested our implementation described in section IV, we noticed some unwanted behavior in building the trees. Even after adjusting the parameters, including the minimum and maximum Gini index and the maximum number of samples in a leaf, the problems persisted. In particular we noticed three issues:

1) a low number of splits per Decision Tree,
2) the Gini index kept increasing in successive splits and
3) according to the distance matrix, the all the scenarios were very different to each other.

The average number of splits was five on a dataset containing around 7000 samples, filtered according to minimum distance headway. This is in our view no where near sufficient to properly separate different driving scenario types, fact confirmed by the distance matrix. The minimum distance between any two samples was around **??**, which made any clustering attempt

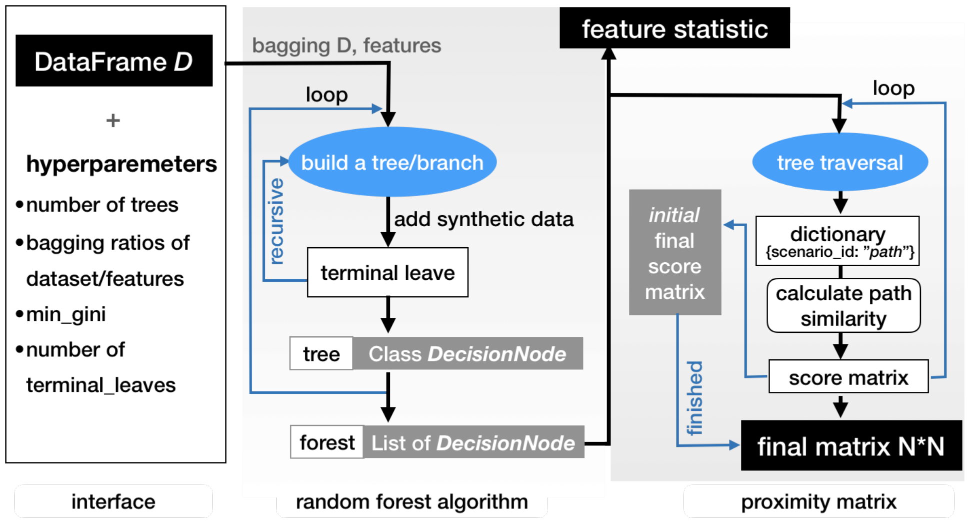


Fig. 14: An Overview of the Implementation from dataset to proximity matrix

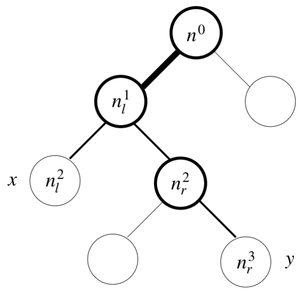


Fig. 15: Example of the proximity path from a pair (x, y) where x located in leave node n_l^2 while y in n_r^3. $p_x = \{n^0, n_l^1, n_l^2\}$, $p_y = \{n^0, n_l^1, n_r^2, n_r^3\}$. Thus in this tree $score_{x,y} = \frac{2}{5}$

pointless. Furthermore, we noticed the Gini index steadily increasing with the depth of the tree, which is counter-intuitive, as nodes at a bigger depth should be purer.

B. Improvements

In order to address these issues, we updated our URF implementation in a number ways, inspired by [6]:

1) Replacing Gini index with Gini gain
2) Adding multiple distributions for the synthetic data
3) Flexible number of synthetic points per node
4) Introducing new parameters

We will tackle each improvement step individually and explain the role it plays in the algorithm.

1) Gini Gain: We base our splitting decision on the Gini gain instead of the Gini index as this assures a continuous improvement in the quality of our splits. Coupled with the problems related to the Gini index mentioned in section V-A and the fact that Scikit-learn [17] is also removing the option to stop RF growth via Gini index, it seems that this form of choosing a split points is not suited for unsupervised learning.

$$\Delta R(t, t_L, t_R) = r(t) - \frac{M(t_L)}{M(t)} r(t_L) - \frac{M(t_R)}{M(t)} r(t_R) \quad (1)$$

As in [6], we used the standard formulation of the Gini gain (1), where $\Delta R(t, t_L, t_R)$ is the Gini gain obtained from splitting node t into child nodes t_L and t_R, $r(t)$ is the Gini index and $M(t)$ is the number of data points of node t. This links the current purity of the node with that of its children, leading to a relative measure for improvement as opposed to the absolute measure we had before. We generate new synthetic points at each depth, which means that a node which is evaluated for a split will not contain the same synthetic data points as when it was created using the synthetic points from its parent. Furthermore, when we find a cluster with evenly spread out points, the Gini index will approach its maximum of 0.5. This explains the phenomenon of ever increasing Gini indices observed in section V-A. We conclude that the Gini gain is more appropriate in an unsupervised scenario.

2) Multiple distributions: Next, we adopt multiple distributions for generating the synthetic points [6]. Each node randomly chooses how to model its synthetic data from a fixed pool of distributions and uses that to generate synthetic points for every dimension. We chose the following distribution PDFs:

$$\text{Unif}(x| \; -3, 3) \quad (2)$$

$$\mathcal{N}(x| \; 0, 1) \quad (3)$$

$$\frac{1}{2}(\mathcal{N}(x| \; -1, 0.7^2) + \mathcal{N}(x| \; 1, 0.7^2)) \quad (4)$$

One can notice that all of them have the majority of their points between -3 and 3. As such, we transform our data to an interval of length six around zero (not necessarily centered at zero), by subtracting the mean and scaling by the difference

between the smallest and largest data point of that node.
We noticed that the type of distribution has a major effect on the position of the splits. For the normally distributed points, the splits were balanced, meaning that the *smaller* child node (the one that received less data points) still secured at least 15% of the data. However, when looking at the Uniform or the Gaussian mixture, we observed a big tendency of the smaller child receiving one or even zero data points. This might become clear from Figure 16, where we achieve the best Gini gain by splitting at -3, for instance. To address this, we have introduced a new parameter to the URF that specifies the minimum number of data points for a child node. This prevents unbalanced splits but may have a negative impact if set too high, as we might disregard small clusters at the edges of the range.
By allowing for different distribution we eliminate any bias that a particular distribution might bring. The list of distributions can, of course, be extended.

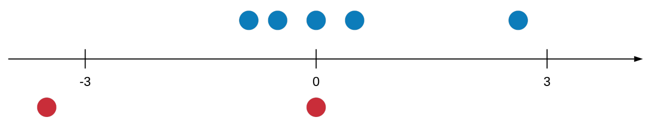


Fig. 16: Unwanted split at -3. Blue points represent actual data, red points synthetic data

3) Flexible amount of synthetic data: Instead of having exactly as much synthetic as real data in every node, we decide to allow for a certain flexibility [6]. The number of synthetic points are still connected to the number of real data points, but the ration may vary. We calculate based on equation 5, where the $M_{S,l}(z)$ and $M_{S,r}(z)$ represent the number of synthetic points of the left and right child of t respectively, assuming a split at z. $M_D(t)$ is the number of real data points of node t and $\Pr_t$ is the CDF of the distribution chosen for node t.

$$\begin{aligned} M_{S,l}(z) &= M_D(t)\Pr_t(Z \leq z) \\ M_{S,r}(z) &= M_D(t) - M_{S,l}(z) \end{aligned} \tag{5}$$

The number of synthetic points that node t passes to its children still matches the number of data points, but they are not equally divided. The reasoning is that, if at some point we split a valid cluster close to the edges of the interval, the corresponding node will get few synthetic points and thus be almost pure. This discourages subsequent splits of that node as the Gini gain will be marginal if at all.

The effect of all these changes is that we have increased the average number of splits per DT from 5 to 15. When looking at the features picked for splitting, the most frequent ones are consistent over multiple runs which indicates that the URF has indeed found some structure. On the downside, the distance matrix still reports low similarities between any scenarios. This could, however, also be a result of the feature set. Regardless, considering the number of hyper parameters our model has, additional testing is required to conclude what the optimal values are.

VI. Clustering

After generating a distance matrix, we run classical clustering methods on it. Unlike the URF IV, where we needed a custom implementation because of the synthetic data at each split, we use the Scikit-learn library [17] for the clustering algorithms, because of its efficiency and flexibility. Our implementation packages everything in a command line interface which makes it easy to switch between three different clustering techniques and adjust their parameters:

- Hierarchical clustering
- DBSCAN
- Spectral clustering

1) Hierarchical Clustering: Based on [6], we have used Hierarchical clustering on the distance matrix. The advantages of this method of clustering are its intuitiveness and ability to specify the number of clusters. It can also easily generate a cluster map, i.e. a sorted heat map, based on its output. This allows us to decide how many clusters are in the dataset. The clusters appear as squares on the diagonal of the cluster map. Figure 17 exhibits three main clusters, a large one in the middle and two smaller ones above and below.

Hierarchical clustering starts with each data point as being part of its own cluster and then progressively merges the closest clusters in order to group the data. The measure by which the closest clusters are selected in each step can differ. We focus on three metrics:

- *complete* distance
- *single* distance
- *average* distance

In the *complete* case, the distance between two clusters is defined by the maximum distance between one point from the first cluster and another point from the second one. It is less sensitive to outliers but biases towards elliptic clusters. The *single* case is similar, but takes the minimum distance. This makes it more susceptible to outliers but allows for different cluster shapes. The *average* case, as the name implies, takes the average distance between all the points of the two clusters. Regarding its properties, it is a mix of the above two, being less sensitive to outliers than single distance and preferring elliptical clusters.

2) DBSCAN: DBSCAN is a density based approach that has two parameters, a radius *epsilon* and a number of neighbors.

$$\begin{aligned} V(p) &= \{x \in \mathcal{D} | \ \|p - x\| \leq e\}, \ p \in \mathcal{D} \\ \mathcal{C} &= \{p \in \mathcal{D} | \ |V(p)| \geq n\} \\ \mathcal{B} &= \{p \in \mathcal{D} \setminus \mathcal{C} | \ \exists x \in \mathcal{D}, p \in V(x)\} \\ \mathcal{U} &= \mathcal{D} \setminus (\mathcal{C} \cup \mathcal{B}) \end{aligned} \tag{6}$$

Where $\mathcal{D}$ is the set of data points, e is epsilon and n is the number of neighbors. We call $V(p)$ the vicinity of point p. $\mathcal{C}$ is the set of *core* points, $\mathcal{B}$ the set of *border* points and $\mathcal{U}$ the set of *unlabeled* points. Unlabeled points are disregarded and

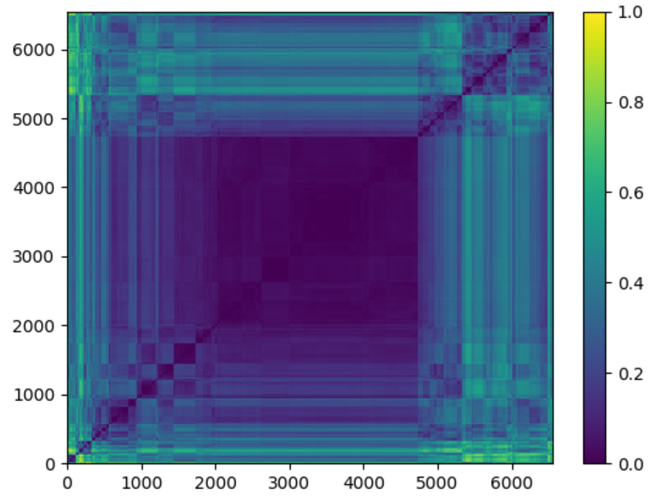


Fig. 17: Cluster map generated from the DHW dataset using the Scikit-learn implementation of RF with 20000 synthetic data points, average distances

are not assigned to any cluster. For core and border points, construct edges to the points in their vicinity, thus obtaining a graph. The connected components of core points form clusters. The border points are assigned to whichever cluster they have more edges.

The advantage of DBSCAN is that it performs well, regardless of the shape of the clusters. It is also very robust to outliers, thanks to the unlabeled points. In theory, this allows for a post filtering of traffic scenarios that were wrongly added to the dataset. A disadvantage of DBSCAN is its sensitivity to the relative densities of different overlapping clusters. Although good on paper, we found it difficult to apply DBSCAN on our distance matrix. The large values in the matrix forces a very narrow window for the radius and number of neighbors, mostly leading to all the points being unlabeled or all of them in the same cluster. Note that DBSCAN does not allow the user to set the number of clusters.

3) Spectral clustering: Spectral clustering works on a wide range on data distributions and is highly flexible. A detailed explanation of algorithm is beyond the scope of this report, but we give a brief overview. The data is first transformed into a graph, by means of k-Nearest neighbors, for instance. The diagonal degree matrix contains the degree of node i on position (i, i). The Laplacian matrix is build by subtracting the adjacency matrix from the degree matrix. Principal component analysis is applied to the Laplacian, reducing the dimensionality. Finally, k-Means is performed on the transformed space.

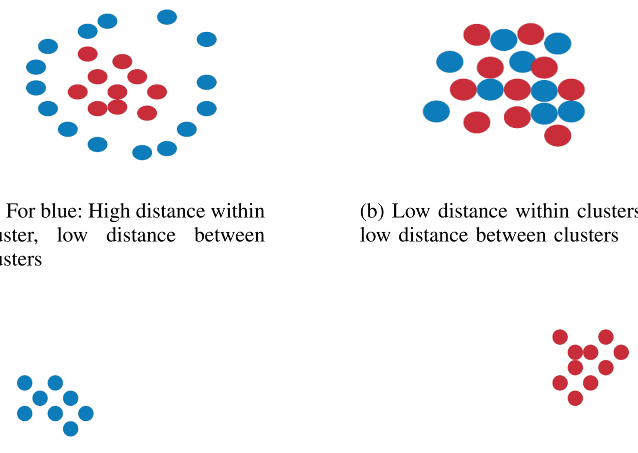
Spectral clustering allows for the number of clusters to be specified and has no restrictions concerning cluster shapes. In contrast to the other two methods, it works on the affinity matrix and not the distance matrix. We compute the affinity matrix by subtracting the distance matrix from an all one matrix. We found that spectral clustering provided similar performance to hierarchical clustering, with respect to the silhouette coefficient, even though the silhouette coefficient prefers globular clusters.

Silhouette coefficient

The silhouette coefficient (SC) provides a metric to evaluate the performance of a clustering algorithm. As we don't have any labels, we can't have a traditional loss function that takes into account the ground truth. The SC takes into account the distance between different clusters as well as the distance inside each cluster.

$$\begin{aligned}
a(i) &= \frac{1}{|C_i|} \sum_{j \in C_i, j \neq i} d(i,j) \\
b(i) &= \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \\
s(i) &= \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1 \\
s(i) &= 1, \text{ if } |C_i| = 1
\end{aligned} \tag{7}$$

$a(i)$ represents the average distance within a cluster, $b(i)$ the average distance to the closest cluster and $s(i)$ is the silhouette coefficient itself, for cluster i. Note that $-1 \leq s(i) \leq 1$. The global SC is the average over all $s(i)$.



(a) For blue: High distance within cluster, low distance between clusters

(b) Low distance within clusters, low distance between clusters

(c) Low distance within clusters, high distance between clusters

Fig. 18: Comparison of different clusterings with respect to the silhouette coefficient. In (a), one cluster encapsulates the other and the silhouette coefficient becomes negative for the blue points. In (b), it is around zero because the clusters overlap. In (c), it approaches 1, as the clustering is desirable.

Figure 18 shows the three cases of the SC, negative, around zero and close to one. Note that Figure 18a might be a desirable clustering by other measures, but the SC gives it a bad score (the global SC is, however, improved by the red group). The SC biases towards globular clusters. Because of this reason, it does not serve as a good comparison between different clustering approaches. It may, however, be used to compare between different parameters for the same method. Another downside is that it does not take into account the number of samples in a particular cluster. This lead, in our experience, to undesirable results, as it could happen that an instance in which one cluster contained most of the samples was rated better than another more balanced instance. This is not to say that we expect the results to be balanced, but the unbalance was so extreme, that most clusters only had one sample.

VII. Evaluation

A. Methodology

We performed tests on all 'Small' datasets, which contained about 6500 to 7000 samples. The forests contained 200 trees, with minimum Gini gain between 0.05 and 0.01. The maximum number of samples in leaf nodes was set to 20, 40 and 60. Note that the splitting may also stop due to insufficient Gini gain. When splitting, we forced a minimum number of 5 or 10 samples to each child, to avoid extremely unbalanced splits. The maximum depth of the trees was capped at 10. For every dataset we performed 4 runs, with 4 different subsets of features (TTC, DHW, THW and all features). For every distance matrix we performed 30 Hierarchical clusterings, 10 for every distance metric (single, average and complete), with the number of clusters ranging from 3 to 12. We ran the URF training and distance matrix calculations on 4 threads. The clustering was performed on a single thread. The entire process took around 4 hours.

B. Results

The best results, judged by the silhouette coefficient (SC), were achieved on the DHW dataset using all the available features. It registered a silhouette coefficient of 0.065 for average distances on three clusters. Upon closer inspection, the cluster map only showed 2 clusters (Figure 20). Another run with 2 clusters saw the SC increase to 0.071.

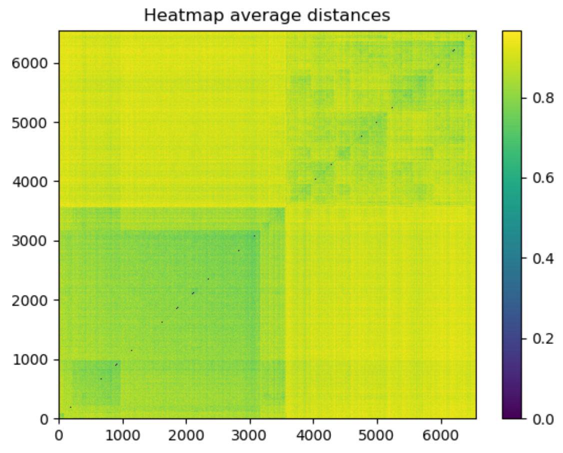


Fig. 19: Cluster map generated from the DHW dataset. Numbers represent distances.

As to be expected from Figure 20, the silhouette coefficient kept decreasing with higher number of clusters. The differences between the various clustering methods and the number of clusters were negligible, all ranging from 0 to 0.07. We can also see from the heat map, that the scenarios recognized as similar at all, with a minimum distance of around 0.6.

By looking at the number of splits performed on each feature, we can say which features have a higher significance. Although not exactly the same among all datasets, the top features have always been related to the acceleration, breaking time or velocity of the ego vehicle at different times. Features regarding neighbors have been used less often.

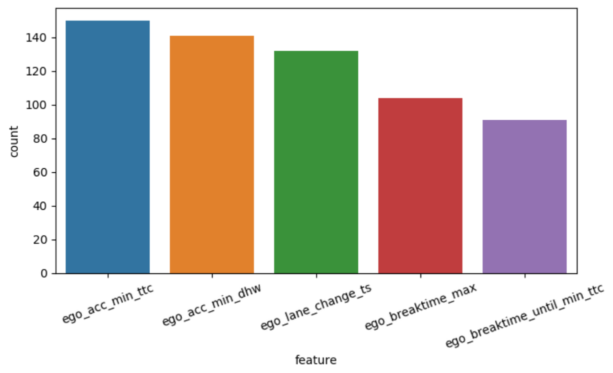


Fig. 20: Most used features on the DHW dataset

Using this information we can plot the values of features for the different clusters. In Figure 21b, for instance, we can see that the ego vehicle in cluster one doesn't brake until the moment of minimum TTC at all, whereas in cluster zero it does. Similarly, the ego vehicle of cluster one does not accelerate at the moment of minimum DHW, whereas it has much higher variance in cluster zero (Figure 21a).

When looking at visualizations of the clusters, we couldn't discern any particularities of the two clusters. We believe this is in part due to the distance matrix but it may be a problem of the features as well. We also ran the library implementation of Random Forests from Scikit-learn, where the scenarios were considered much more similar to another and this yielded a much higher silhouette coefficient (around 0.5). The clusters, however, didn't seem alike to the human eye. Due to memory limitations, 20000 synthetic points were added to the around 7000 samples. This is in no way enough to uniformly cover the entire space and is exactly the reason we opted for a custom implementation of URFs.

VIII. Conclusion and Future Work

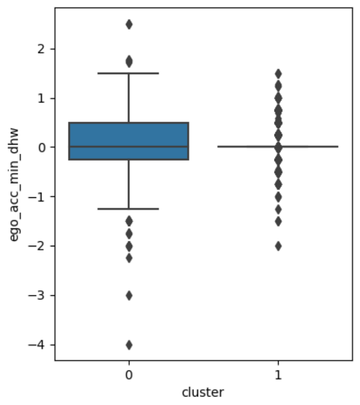
A. Subsection Heading Here

Subsection text here.

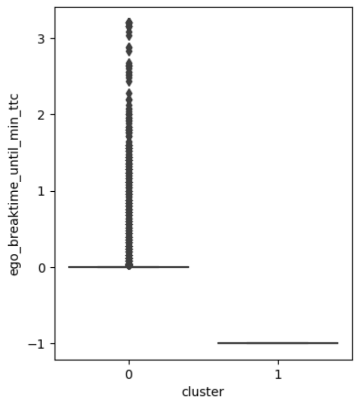
References

[1] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[2] D. Zhao and H. Peng, "From the lab to the street: Solving the challenge of accelerating automated vehicle testing," *arXiv preprint arXiv:1707.04792*, 2017.



(a) Box plot of the acceleration at minimum DHW



(b) Box plot of the brake time until minimum TTC

Fig. 21: Values of the important features (based on the number of splits) in the clusters

[3] R. Matthaeia, A. Reschkaa, J. Riekena, F. Dierkesa, S. Ulbricha, T. Winkleb, and M. Maurera, "Autonomous driving: Technical, legal and social aspects," 2015.

[4] "Pegasus research project," https://www.pegasusprojekt.de, accessed: 2020-01-13.

[5] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 719–726.

[6] F. Kruber, J. Wurst, E. S. Morales, S. Chakraborty, and M. Botsch, "Unsupervised and supervised learning with the random forest algorithm for traffic scenario clustering and classification," *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2463–2470, 2019.

[7] F. Kruber, J. Wurst, and M. Botsch, "An unsupervised random forest clustering technique for automatic traffic scenario categorization," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2811–2818.

[8] J. Halkias and J. Colyar, "Ngsim interstate 80 freeway dataset," *US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA*, 2006.

[9] J. Colyar and J. Halkias, "Us highway 101 dataset," *Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030*, 2007.

[10] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[12] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2118–2125.

[13] "Matlab and python source code to handle the data, and create visualizations for highd dataset," https://github.com/RobertKrajewski/highD-dataset, accessed: 2019-11-13.

[14] M. Benmimoun, F. Fahrenkrog, A. Zlocki, and L. Eckstein, "Incident detection based on vehicle can-data within the large scale field operational test "eurofot"," in *22nd Enhanced Safety of Vehicles Conference (ESV 2011), Washington, DC/USA*, 2011.

[15] T. Shi and S. Horvath, "Unsupervised learning with random forest predictors," *Journal of Computational and Graphical Statistics*, vol. 15, no. 1, pp. 118–138, 2006.

[16] B. Liu, Y. Xia, and P. S. Yu, "Clustering through decision tree construction," in *Proceedings of the ninth international conference on Information and knowledge management*, 2000, pp. 20–29.

[17] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.