Unsupervised Random Forest Learning for Traffic Scenario Categorization

Friedrich Kruber, Jonas Wurst, Michael Botsch, and Samarjit Chakraborty

1 Introduction

Looking at traffic scenarios microscopically, there is an infinite number of scenarios. A somewhat higher visual range shows that they nevertheless follow certain patterns and can be assigned to categories with a high degree of similarity. Testing representatives from each category ensures a broad scope, while minimizing the effort in the validation process. In order to perform a relevance evaluation, one has to memorize and structure traffic scenarios. Therefore, the vast amount of sensor data needs to be shrinked. This can be achieved by representing a traffic situation with a set of relevant features. These features can then be used in machine learning algorithms for analysis purposes.

A training dataset of recorded traffic scenarios is usually manually labeled in order to run supervised classification algorithms. In contrast to that, unsupervised learning yields to identify patterns in datasets, where the availability of labels for training machine learning models is absent. The main focus here is the introduction of an unsupervised learning procedure for the categorization of traffic scenarios, only given the input from arbitrary data sources.

The chapter is organized as follows. After the methodological introduction into Decision Trees and Random Forests in Sects. 2 and 3, the method is extended for its usage in the field of unsupervised learning in Sect. 4. Its application for the unsupervised clustering of real world traffic scenarios is discussed in Sect. 5. Finally, the versatility of Random Forests is demonstrated by another method, which

F. Kruber · J. Wurst · M. Botsch
Technische Hochschule Ingolstadt, Ingolstadt, Germany
e-mail: friedrich.kruber@thi.de; jonas.wurst@thi.de; michael.botsch@thi.de

S. Chakraborty (✉)
UNC Chapel Hill, Chapel Hill, NC, USA
e-mail: samarjit@cs.unc.edu

V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_20

integrates a Random Forest into a Deep Learning architecture to tackle the problem of Open-Set recognition for traffic scenarios.

Before moving to the next section about Decision Trees, some notations will be introduced. A dataset $\mathcal{D}$ consists of M datapoints $\boldsymbol{x}_m \in \mathbb{R}^N$, referred to as feature vectors. Supervised learning requires a training dataset containing a target vector y_m for each feature vector $\boldsymbol{x}_m$. Assuming that the possible output is a scalar it yields

$$\mathcal{D}_\mathrm{s} = \left\{(\boldsymbol{x}_1, y_1), \ldots, \left(\boldsymbol{x}_{M_\mathrm{s}}, y_{M_\mathrm{s}}\right)\right\}. \tag{1}$$

Contrary to that, in unsupervised learning the dataset does not provide any information about the objective values. Therefore, those datasets are defined as

$$\mathcal{D}_\mathrm{u} = \left\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M_\mathrm{u}}\right\}. \tag{2}$$

The feature vector $\boldsymbol{x}$ as well as the target y are not deterministic. Therefore, the feature vector $\boldsymbol{x}$ is a realization of the random variable x and y a realization of the random variable y.

Supervised machine learning aims to find a function f based on $\mathcal{D}_\mathrm{s}$, which performs the mapping from the input variable x to the target y. Depending on the target value characteristics, supervised learning can be in the form of *classification* or *regression*. If y is of categorical type, the function is called classification. With $\mathsf{x} \in \mathbb{R}^N$, the classification is defined as

$$f : \mathbb{R}^N \to \{c_1, \ldots, c_K\}, \mathsf{x} \mapsto \hat{\mathsf{y}}, \tag{3}$$

where $\hat{\mathsf{y}} \in \{c_1, \ldots, c_K\}$ is the categorical predicted output. If the output is continuous, i.e. $\hat{\mathsf{y}} \in \mathbb{R}$, the function is called regression:

$$f : \mathbb{R}^N \to \mathbb{R}, \mathsf{x} \mapsto \hat{\mathsf{y}}. \tag{4}$$

The aim of all supervised learning methods is to find a function f, which gains the highest performance. Therefore, the performance measurement called risk

$$R(f) = \underbrace{\mathrm{E}_{\mathsf{x},\mathsf{y}}\left\{\mathcal{L}(\mathsf{y}, f(\mathsf{x}))\right\}}_{\text{expectation of } \mathcal{L}} = \int_{\mathbb{R}^N} \sum_{k=1}^{K} \mathcal{L}(c_k, f(\boldsymbol{x}))\, \underbrace{\mathrm{p}(\mathsf{x} = \boldsymbol{x}, \mathsf{y} = c_k)}_{\text{joint probability density function}}\, d\boldsymbol{x} \tag{5}$$

$$R(f) = \mathrm{E}_{\mathsf{x},\mathsf{y}}\mathcal{L}(\mathsf{y}, f(\mathsf{x})) = \int\limits_{\mathbb{R}^N} \int\limits_{\mathbb{R}} \mathcal{L}(y, f(\boldsymbol{x}))\, \mathrm{p}(\mathsf{x} = \boldsymbol{x}, \mathsf{y} = y)\, dy d\boldsymbol{x}, \tag{6}$$

for classification and regression is introduced. $\mathcal{L}$ denotes the loss and E the expectation. The goal is to find a function f_B, which gains the highest performance by minimizing $R(f)$

$$f_{\mathrm{B}} = \arg\min_{f}\{R\,(f)\}, \tag{7}$$

where f_{B} is known as the Bayes classifier or Bayes regression function. The density functions are usually unknown. Instead, risk can be estimated through the empirical risk by employing the dataset $\mathcal{D}_{\mathrm{s}}$,

$$R_{\mathrm{emp}}\,(f, \mathcal{D}_{\mathrm{s}}) = \frac{1}{M_{\mathrm{s}}}\sum_{m=1}^{M_{\mathrm{s}}} \mathcal{L}\,(y_m, f\,(\boldsymbol{x}_m))\,. \tag{8}$$

Various approaches try to minimize the empirical risk in order to find a good mapping f. The approaches differ in their architecture and hence the realized function. In the following, we focus on the proven-in-use ensemble method termed Random Forests, which is constructed from a set of Classification and Regression trees.

2 Classification and Regression Trees

Classification and Regression Trees (CART) have several benefical properties for real world applications. They can model relations between an input $\mathbf{x}$ and the output y independent of the number of features or dataset size. The input variables can be either categorial or ordered, and even both types may be apparent in the feature set. Probably the most important aspect to favor decision trees over many other methods, is the interpretability. All decisions and interactions among features can be interpreted by humans, and thus provide the white-box character of CARTs.

The CART algorithm, introduced in [8], is a specific form of binary decision trees. As depicted in Fig. 1 and explained in detail in what follows, a binary tree can be thought of a multi decision process, as well as a directed graph. In an algorithmic view, binary trees consist of many *if/else* queries. The intuition behind trees is simple, yet understanding the underlying principles is mandatory since they build the basis for the more sophisticated Random Forest algorithms.

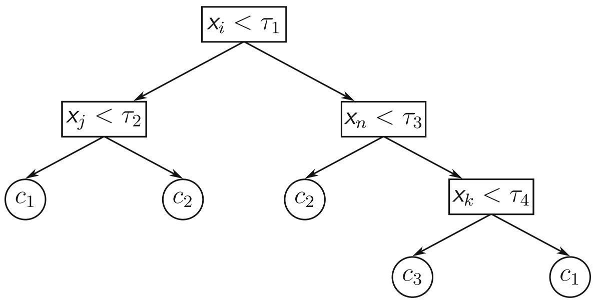

Fig. 1 Example of a binary classification tree

Before diving into the methodology, a set of definitions considering a binary tree is established first. A binary tree $\mathfrak{T}$ is constructed with *nodes* t, which are connected by *edges*. The top node is designated as the *root*, such that all edges are directed downwards from the root. Two nodes are connected by a single edge. The information flows from the parent node to the child node. In a binary tree, intermediate nodes have exactly two children, but only one parent node. If a node does not possess children, it is termed as *terminal node*, or *leaf*.

Starting from the input space $\mathfrak{X} \in \mathbb{R}^N$, at each *if/else* decision stage the input is split into two disjoint subspaces $\mathfrak{X}_i$ and $\mathfrak{X}_j$ of $\mathbb{R}^N$ for which $\mathrm{p}(\mathbf{x} \in \mathrm{t}) > 0$. These subspaces are represented by the two children nodes. When propagating through all stages of a tree, each datapoint $\mathbf{x}_m$ is assigned to a constant prediction $\hat{y}$ within that subspace. This subspace is corresponding to the terminal node in the tree.

2.1 Computing the Optimal Split

The aim of growing a tree is to minimize the risk according to Eq. (5) and Eq. (6) for classification or regression problems, respectively. This is achieved by finding the best *splits* and will be derived in the following.

We define $\tilde{\mathfrak{T}}$ as the subspace of $\mathfrak{X}$ representing all terminal nodes of a tree. The function, which assigns each input $\mathbf{x}$ to a terminal node $\mathrm{t} \in \tilde{\mathfrak{T}}$ can be formalized as

$$\rho : \mathfrak{X} \to \tilde{\mathfrak{T}}, \mathbf{x} \mapsto \mathrm{t}. \tag{9}$$

Given an assignment function $\upsilon(t) \in \{c_1, \ldots, c_K\}$ for classification, or $\upsilon(t) \in \mathbb{R}$ for regression, a mapping f corresponds to $\tilde{\mathfrak{T}}$ so that $f(\mathbf{x}) = \upsilon(\mathrm{t})$ for all inputs $\mathbf{x} \in \mathrm{t}$, then

$$f(\mathbf{x}) = \upsilon\left(\rho(\mathbf{x})\right). \tag{10}$$

Based on the previous equations, one obtains the risk of the mapping realized as

$$R(f) = \int_{\mathfrak{X}} \mathrm{E}_{y|x}\left\{\mathcal{L}\left(y, f\left(x\right)\right)\right\} \mathrm{p}\left(\mathbf{x} = x\right) dx \tag{11}$$

$$= \sum_{\mathrm{t} \in \tilde{T}} \mathrm{E}_{y|x \in \mathrm{t}}\left\{\mathcal{L}\left(y, \upsilon\left(\mathrm{t}\right)\right)\right\} \mathrm{p}\left(\mathbf{x} \in \mathrm{t}\right). \tag{12}$$

The global error in Eq. (11) is equal to sum of the local errors within all terminal nodes, Eq. (12). Hence, the risk can be minimized locally with

$$r_{\min}(\mathrm{t}) = \min_{\upsilon}\left\{\mathrm{E}_{y|x \in \mathrm{t}}\left\{\mathcal{L}\left(y, \upsilon\left(\mathrm{t}\right)\right)\right\}\right\}, \tag{13}$$

leading to the overall minimum prediction risk for $\tilde{\mathfrak{T}}$ of $\mathbb{R}^N$

$$R_{\min}(\tilde{\mathfrak{T}}) = \sum_{\tilde{\mathfrak{T}}} r_{\min}(\mathrm{t})\mathrm{p}(\mathbf{x} \in \mathrm{t}). \tag{14}$$

The local minimum risk for regression in a node $\mathrm{t} \in \tilde{\mathfrak{T}}$ is

$$r_{\min}(\mathrm{t}) = \hat{\sigma}^2_{\tilde{\mathfrak{T}}}. \tag{15}$$

For classification, the minimum local risk is obtained with the class of highest posterior probability

$$r_{\min}(\mathrm{t}) = \min_{c_l} \left\{ \sum_{k=1}^{K} (1 - \delta(c_k, c_l))\mathrm{p}(y = c_k|\mathbf{x} \in \mathrm{t}) \right\} \tag{16}$$

$$= 1 - \max_{c_k} \{\mathrm{p}(y = c_k|\mathbf{x} \in \mathrm{t})\}, \tag{17}$$

where $\delta(\cdot, \cdot)$ is one if both arguments are equal, otherwise zero. Typically, classification is realized through a class probability estimation, which leads to the local risk in a node $\mathrm{t} \in \tilde{\mathfrak{T}}$ as

$$r_{\min}(\mathrm{t}) = \sum_{k=1}^{K} \mathrm{p}(y = c_k|\mathbf{x} \in \mathrm{t})(1 - \mathrm{p}(y = c_k|\mathbf{x} \in \mathrm{t})). \tag{18}$$

When growing a tree, the aim is to reduce the local risk as best as possible with an optimal split $\mathfrak{s}_{\mathrm{opt}}$ in order to divide a node $\mathrm{t} \in \tilde{\mathfrak{T}}$ into a left $\mathrm{t_L}$ and right $\mathrm{t_R}$ childnode. We define this as a new fraction of $\tilde{\mathfrak{T}}'$ of $\mathbb{R}^N$. A split reduces the risk with

$$\Delta R(\mathfrak{s}, \mathrm{t}) = R_{\min}(\tilde{\mathfrak{T}}) - R_{\min}(\tilde{\mathfrak{T}}') \tag{19}$$

$$= \mathrm{p}(\mathbf{x} \in \mathrm{t})(r_{\min}(\mathrm{t}) - \mathrm{p}(\mathbf{x} \in \mathrm{t_L}|\mathbf{x} \in \mathrm{t})r_{\min}(\mathrm{t_L}) - \mathrm{p}(\mathbf{x} \in \mathrm{t_R}|\mathbf{x} \in \mathrm{t})r_{\min}(\mathrm{t_R})), \tag{20}$$

and the relative risk reduction is

$$\Delta r(\mathfrak{s}|\mathrm{t}) = \frac{R_{\min}(\tilde{\mathfrak{T}}) - R_{\min}(\tilde{\mathfrak{T}}')}{\mathrm{p}(\mathbf{x} \in \mathrm{t})} \tag{21}$$

$$= r_{\min}(\mathrm{t}) - \mathrm{p}(\mathbf{x} \in \mathrm{t_L}|\mathbf{x} \in \mathrm{t})r_{\min}(\mathrm{t_L}) - \mathrm{p}(\mathbf{x} \in \mathrm{t_R}|\mathbf{x} \in \mathrm{t})r_{\min}(\mathrm{t_R}). \tag{22}$$

Now, the best split $\mathfrak{s}_{\mathrm{opt}}$ for a node t is achieved by maximizing $\Delta r(\mathfrak{s}|\mathrm{t})$

$$\mathfrak{s}_{\mathrm{opt}}(\mathrm{t}) = \operatorname*{argmax}_{\tilde{\mathfrak{s}}} \left\{\Delta r(\tilde{\mathfrak{s}}|\mathrm{t})\right\}. \tag{23}$$

In practice the required probability density functions are not known. Therefore, in the following we consider how to perform a split with a given dataset $\mathcal{D}_\mathrm{s}$.

2.2 Growing a Tree

Since the probability function $\mathrm{p}(\mathbf{x} \in \mathrm{t})$ is unknown, it has to be approximated with a training dataset $\mathcal{D}_\mathrm{s}$ containing M datapoints. We define a set $\mathfrak{M}(\mathrm{t}) = \{m \in \mathfrak{M} : \boldsymbol{x}_m \in \mathrm{t}\}$, which contains all datapoints of $\mathcal{D}_\mathrm{s}$ in the node t. The empirical estimate $\hat{\mathrm{p}}$ can then be computed with

$$\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t}) = \frac{|\mathfrak{M}(\mathrm{t})|}{M} = \frac{M(\mathrm{t})}{M}, \tag{24}$$

with the assumption that $\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t}) > 0$ for all $\mathrm{t} \in \tilde{\mathfrak{T}}$. Now, we can reformulate Eq. (13) and Eq. (14)

$$\hat{r}_{\min}(\mathrm{t}) = \min_{\upsilon} \frac{\sum_{\mathfrak{M}(\mathrm{t})} \mathcal{L}(y_m, \upsilon(\rho(\boldsymbol{x}_m)))}{M(\mathrm{t})}, \text{ and} \tag{25}$$

$$\hat{R}_{\min}(f) = \sum_{\tilde{\mathfrak{T}}} \hat{r}_{\min}(\mathrm{t})\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t}). \tag{26}$$

For regression we set $\upsilon = \hat{\mu}(\mathrm{t})$, so that $\hat{r}_{\min}(\mathrm{t}) = \hat{\sigma}^2_{\tilde{\mathfrak{T}}}(\mathrm{t})$, with μ and σ^2 denoting the expectation and variance.

For classification the estimate for $\mathrm{p}(y = c_k|\mathbf{x} \in \mathrm{t}), k = 1, \ldots, K$ has to be calculated to compute $\hat{r}_{\min}(\mathrm{t})$. We define another set $\mathfrak{M}_k(\mathrm{t}) = \{m \in \mathfrak{M} : \boldsymbol{x}_m \in \mathrm{t} \text{ and } y_m = c_k\}$ for all datapoints in the node t belong to a class c_k, so that the empirical probability estimation is

$$\hat{\mathrm{p}}(y = c_k|\mathbf{x} \in \mathrm{t}) = \frac{M_k(\mathrm{t})}{M(\mathrm{t})}. \tag{27}$$

Hence, Eq. (18) turns into

$$\hat{r}_{\min}(\mathrm{t}) = \sum_{k=1}^{K} \hat{\mathrm{p}}(y = c_k|\mathbf{x} \in \mathrm{t})(1 - \hat{\mathrm{p}}(y = c_k|\mathbf{x} \in \mathrm{t})) \tag{28}$$

$$= \sum_{k=1}^{K} \sum_{\substack{\mathrm{k'}=1 \\ k' \neq k}}^{K} \hat{\mathrm{p}}(y = c_k|\mathbf{x} \in \mathrm{t})\hat{\mathrm{p}}(y = c_{k'}|\mathbf{x} \in \mathrm{t}), \tag{29}$$

and Eq. (17) turns into

$$\hat{r}_{\min}(\mathrm{t}) = \min_{c_l} \left\{ \sum_k (1 - \delta(c_k, c_l)) \hat{\mathrm{p}}(y = c_k | \mathbf{x} \in \mathrm{t}) \right\} \tag{30}$$

$$= 1 - \max_{c_k} \left\{ \hat{\mathrm{p}}(y = c_k | \mathbf{x} \in \mathrm{t}) \right\}. \tag{31}$$

In order to compute the best split, we need to formulate the empirical relative risk reduction, which is

$$\Delta\hat{r}(\mathfrak{s}|\mathrm{t}) = \hat{r}_{\min}(\mathrm{t}) - \hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_L} | \mathbf{x} \in \mathrm{t}) \hat{r}_{\min}(\mathrm{t_L}) - \hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_R} | \mathbf{x} \in \mathrm{t}) \hat{r}_{\min}(\mathrm{t_R}), \tag{32}$$

where

$$\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_L} | \mathbf{x} \in \mathrm{t}) = \frac{\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_L})}{\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t})} = \frac{M(\mathrm{t_L})}{M(\mathrm{t})}, \tag{33}$$

and

$$\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_R} | \mathbf{x} \in \mathrm{t}) = \frac{\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t_R})}{\hat{\mathrm{p}}(\mathbf{x} \in \mathrm{t})} = \frac{M(\mathrm{t_R})}{M(\mathrm{t})}. \tag{34}$$

Now, that all parts for $\Delta\hat{r}(\mathfrak{s}|\mathrm{t})$ are defined, the optimal empirical split at a node t can be computed with

$$\hat{\mathfrak{s}}_{\mathrm{opt}}(\mathrm{t}) = \underset{\tilde{\mathfrak{s}}}{\mathrm{argmax}} \left\{ \Delta\hat{r}(\tilde{\mathfrak{s}}|\mathrm{t}) \right\}. \tag{35}$$

Note, that the border of the partition of t is a hyperplane perpendicular to one of the axes of $\mathbf{x}_n$. The evaluation to set the threshold for the split along a feature n can be conducted with a brute-force approach. If all datapoints in t are distinct with respect to feature n, $M(\mathrm{t}) - 1$ splits for the n-th feature have to be evaluated in order to find the optimal split.

Equation. (29) computes the relative risk reduction $\hat{r}_{\min}(\mathrm{t})$ for classification, known as the *Gini impurity* $i(\mathrm{t})$. The purity gain due to a split can be formulated as

$$\Delta i(\mathfrak{s}, \mathrm{t}) = i(\mathrm{t}) - \frac{M(\mathrm{t_L})}{M(\mathrm{t})} i(\mathrm{t_L}) - \frac{M(\mathrm{t_R})}{M(\mathrm{t})} i(\mathrm{t_R}), \tag{36}$$

such that the optimization task turns into

$$\hat{\mathfrak{s}}_{\mathrm{opt}}(\mathrm{t}) = \underset{\tilde{\mathfrak{s}}}{\mathrm{argmax}} \left\{ \Delta i(\tilde{\mathfrak{s}}, \mathrm{t}) \right\}. \tag{37}$$

Growing a tree until the impurity in terminal nodes becomes $i(\mathrm{t}) = 0$, so that all datapoints belong to a single class, is likely to cause overfitting on the training dataset. On contrary, if the growing process is abrupted too early, the subspaces might not be defined well enough to separate the classes properly. In practice, a tree is first fully grown and then *pruned*. The goal is to prune those nodes, which only have a minor effect on the estimated risk. Several heuristics can be applied as pruning criterias. For example, one can define a minimum number of $M(\mathrm{t})$ samples per terminal node. Another criterion is to define a maximum tree depth to reduce complexity. Lastly, a threshold for the minimum purity gain, provided by a split, can be defined as pruning criterion. Tuning these parameters appropriately is task-specific and should be monitored with a validation dataset.

After pruning, assigning a class to a terminal node is achieved by choosing the class with the highest estimated probability $\hat{\mathrm{p}}(y = c_k|\mathbf{x} \in \mathrm{t})$.

3 Ensemble Learning with Random Forests

A Random Forest is a randomized ensemble learning method, which uses a set of binary trees as base learners. Before addressing Random Forests, we will first take a brief look at the concepts of ensemble learning.

3.1 Ensemble Learning

Ensemble Learning methods use several base learning models and combine the predictions of each individual learner with the aim to improve the final prediction. In case of the Random Forest algorithm, each tree is grown independently of the other trees forming the ensemble. The final prediction of the ensemble method is computed as an average or majority vote of all independently constructed predictors.

The advantage of ensemble methods will be illustrated with a simple example based on the principle of *collective wisdom*. For our example, we assume two possible outputs, where one of the two is the correct answer. All voters predict independently of each other with the same error rate ϵ. Each voter is considered to be competent, i. e. the probability of a false prediction is $\mathrm{p}(\epsilon) < 0.5$ [14]. Under these assumptions, the error limit is going towards zero for an infinite number of voters B

$$\lim_{B \to \inf} \epsilon(B) = 0. \tag{38}$$

Given these restrictions, a voter can be interpreted as a Bernoulli variable with $\mu = \epsilon$ and $\sigma = \epsilon(1 - \epsilon)$. The error rate for majority voting can then be calculated using the binominal distribution

$$\epsilon(B) = \sum_{b=\mathfrak{b}}^{B} \binom{B}{b} \epsilon^b (1-\epsilon)^{B-b} \quad \text{with} \quad \mathfrak{b} = \left\lfloor \frac{B}{2} \right\rfloor + 1. \tag{39}$$

In our example we assume $B = 50$ predictors and an individual error rate of $\epsilon = 0.35$ for all predictors. The ensemble error rate for this majority vote is then

$$\epsilon_{\text{ens}} = \sum_{b=26}^{50} \binom{50}{b} \epsilon^b (1-\epsilon)^{50-b} = 0.01 < \epsilon. \tag{40}$$

Although in practice the predictor models are not completely independent, alone due to the shared training dataset, this examples illustrates the benefits of ensemble methods. Generally, the predictive error is composed of the *bias* and *variance* components. The key behind the success of ensemble methods is related to the reduction of the variance component. Ensembles work effectively as long as the bias and correlation of the base learners is low. CARTs, for example, have a small bias but large variance. A set of CARTs reduce variance, and by inducing randomization techniques, the correlation between all base learners can be reduced as described in the next section. Interested readers are referred to [11] for a detailed explanation of the *bias-variance decomposition*.

Following [10], three fundamental reasons explain why ensembles perform well. The first reason is statistical. A learning algorithm tries to identify the best hypothesis in space. When the amount of training data available is too small, the algorithm can find many different hypotheses of similar accuracy. By averaging the hypotheses one can find a good approximation for f by avoid the risk of choosing a wrong hypothesis. The second reason is computational. Finding a split to grow decision trees is conducted in a brute-force manner. Running the local search from many different starting points often provides a better approximation to the true unknown function. The third reason is representational. Given a finite training dataset, none of the candidate models is able to find the true function. Due to the limited dataset, will explore only a finite set of hypotheses and stop searching when the hypothesis fits the training data. By combining several learners to an ensemble, it can be possible to expand the space of representable functions.

Figure 2 illustrates the potential performance gain of ensemble methods. The spiral training dataset $\mathcal{D}_{\text{t},0}$ consists of $M_{\text{t},0} = 20{,}000$ and the test dataset of $M_{\text{v}} = 500$ datapoints. Additional noise is added to the test dataset to increase the difficulty of the two-class classification task. The first two columns depict the training and test dataset, respectively. The two plots on the right-hand side depict the classification performance of a CART ($B = 1$) and a Random Forest constructed with $B = 10$ tree models. In the first row, the classification performance for both methods is comparable due to the relatively large dataset for the problem to be solved. The ensemble reduces the error rate ϵ by 1%. In the second and third row, the number of training datapoints is being reduced to $M_{\text{t},1} = 2000$ and $M_{\text{t},2} = 200$, where $\mathcal{D}_{\text{t},2} \subset \mathcal{D}_{\text{t},1} \subset \mathcal{D}_{\text{t},0}$. With a decreasing training dataset size, the gap

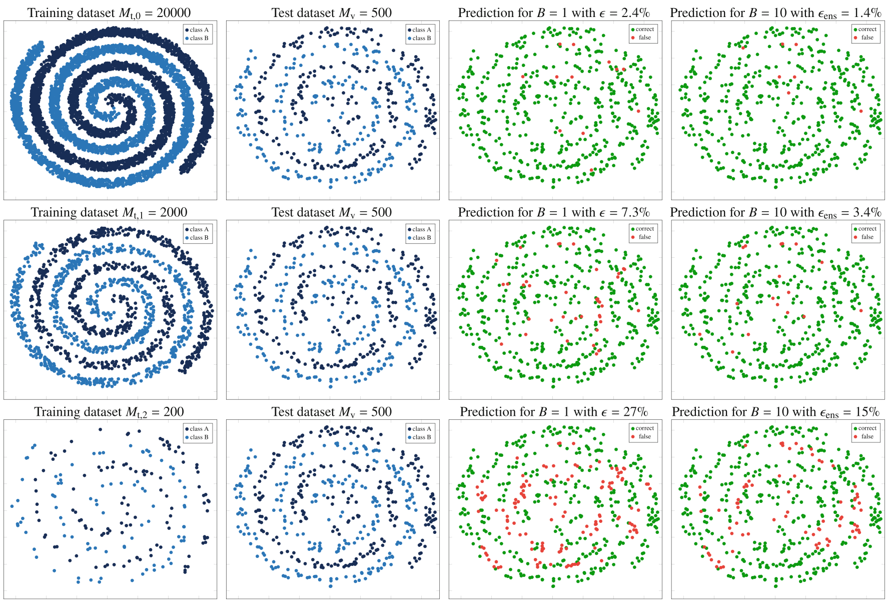

Fig. 2 Classification example on a spiral dataset. The error prediction gap between a single CART model with $B = 1$ and an ensemble method with $B = 10$ learners increases considerably, when the size of the training dataset is a limiting factor for the task to be solved

between a single CART and the ensemble method increases to 3.9% for $M_{t,1}$, and a considerable gain of 12% for $M_{t,2}$. The number of samples required for a good classification performance certainly depends on the difficulty of the problem to be solved. In practice, however, the dataset is often a limiting factor, making ensemble methods appealing.

3.2 Random Forests

The Random Forest algorithm [6] constructs a set of several individual CART as base learners, where each tree is grown independently. After growing the trees, the final ensemble prediction is made by taking the average over the predictions of all trees for regression problems, or by majority vote for classification. Since the Random Forest is composed of CARTs, it inherits the advantages of CARTs, such as the robustness to outliers and noise. The Random Forest is able to handle ordered and categorial variables. It is able to perform a prediction, even when some entries in the input data are missing. Interpretability, though, is not inherited due to the averaging approach for its prediction.

Let a Random Forest be the collection of B trees, where a tree is expressed by $\{\mathfrak{T}_b(\mathbf{x}, \boldsymbol{\theta}_b)\}$. $\boldsymbol{\theta}_b$ is an independent identically distributed random vector. In order to obtain a low variance, the trees should be as uncorrelated as possible. Therefore, the learning procedure is perturbed by two elements forming the vector $\boldsymbol{\theta}_b$, which largely define the characteristics of the Random Forest algorithm. The first element is *bagging* [5], a bootstrap aggregating technique. This means that B new bootstrapped datasets $\mathcal{D}_{\mathrm{s},b}$, $b = 1, \ldots, B$ are randomly drawn by sampling M_{S} datapoints with replacement from $\mathcal{D}_{\mathrm{s}}$. On average, around 37% of the datapoints in $\mathcal{D}_{\mathrm{s}}$ are omitted in each bootstrapped set $\mathcal{D}_{\mathrm{s},b}$

$$\lim_{M_{\mathrm{S}} \to \inf} \left(1 - \frac{1}{M_{\mathrm{S}}}\right)^{M_{\mathrm{S}}} = \frac{1}{e} \approx 0.368. \tag{41}$$

Bagging constructs individual trees by learning with a different dataset $\mathcal{D}_{\mathrm{s},b}$. The second element defines a rule, how the process of splitting a node is conducted. Instead of searching the best split over all N features, only a subset of N_{RF} features is randomly chosen at each node, where $N_{\mathrm{RF}} < N$ holds. A common choice is $N_{\mathrm{RF}} = \left\lceil \sqrt{N} \right\rceil$. Limitating the list of candidates with accelerates the learning procedure as well.

Applying both strategies, bagging and the random subset of features for splitting a node, generate individual base learners and make the Random Forest algorithm likely to benefit from the averaging process. As shown in [6], a Random Forest does not overfit. Therefore, increasing the number of base learners decreases the generalization error, which converges to a limiting value. Another beneficial property of Random Forests and bagging in particular is, that the construction of the base learners is performed with bootstrap samples. The omitted samples of $\mathcal{D}_{\mathrm{s}}$ enable an unbiased estimate of the generalization error during the building process. This is denoted as out-of-bag estimates (oob).

3.2.1 Out-of-Bag Estimates

Bagging allows an unbiased estimate of the generalization error while constructing the ensemble of trees. It comes for free and it can replace an additional validation dataset. As shown in Eq. (41), approximately 37% of all datapoints in $\mathcal{D}_{\mathrm{s}}$ are not part of the bootstrap set $\mathcal{D}_{\mathrm{s},b}$. We denote the trees, which did not use a certain datapoint $\{\boldsymbol{x}_m, y_m\}$, as the set $\mathfrak{B}_m$. Furthermore, we define the out-of-bag class probability estimator as

$$\boldsymbol{f}_{\mathrm{oob}}(\boldsymbol{x}_m, \boldsymbol{\theta}) = \frac{1}{|\mathfrak{B}_m|} \sum_{b \in \mathfrak{B}_m} \boldsymbol{f}_b(\boldsymbol{x}_m, \boldsymbol{\theta}_b), \tag{42}$$

with $\boldsymbol{f}_b(\boldsymbol{x}_m, \boldsymbol{\theta}_b) = \left[\hat{\mathrm{p}}(y = c_1 | \mathbf{x} \in \mathrm{t}_{\mathfrak{T}_b}), \ldots, y = c_K | \mathbf{x} \in \mathrm{t}_{\mathfrak{T}_b})\right]^{\mathrm{T}}$ the vector of all K class probability estimates of the b-th tree. Hence, $\boldsymbol{f}_{\mathrm{oob}}(\boldsymbol{x}_m, \boldsymbol{\theta})$ contains the average class probability over all trees. The majority voting is realized by selecting the class with the highest probability in $\boldsymbol{f}_{\mathrm{oob}}(\boldsymbol{x}_m, \boldsymbol{\theta})$ as $f_{\mathrm{oob}}(\boldsymbol{x}_m)$. Then, the oob estimate for the empirical prediction risk is the averaged sum of errors

$$\hat{R}_{\text{oob}}(f) = \frac{1}{M} \sum_{m=1}^{M} \delta(f_{\text{oob}}(\boldsymbol{x}_m), y_m). \tag{43}$$

The oob estimate has two attractive properties. First, it is equivalent to a test dataset [5]. Second, in contrast to the cross-validation technique, the oob estimate is unbiased, if the number of trees in the Random Forest is large enough.

3.2.2 Proximity Measure

The Random Forest algorithm allows to determine the similarity between two datapoints. Unlike other proximity measures such as the Euclidean, Manhattan or Mahalanobis distance, the Random Forest proximity follows a data-adaptive principle, since the trees are grown according to the training dataset. In order to evaluate the similarity between two datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, one observes if both datapoints end in the same terminal node of a tree. In this case, the similarity value is increased by one. This process is repeated over all trees and the final similarity measure is the average similarity over all trees

$$prox(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{B} \sum_{b=1}^{B} \delta(\mathrm{t}_{\mathfrak{T}_b}(\boldsymbol{x}_i), \mathrm{t}_{\mathfrak{T}_b}(\boldsymbol{x}_j)), \tag{44}$$

with $\mathrm{t}_{\mathfrak{T}_b}(\boldsymbol{x}_i)$ denoting the leaf of the b-th CART in which $\boldsymbol{x}_i$ terminates. Following the idea of proximity, the next Section demonstrates how the Random Forest algorithm can be adapted in order to establish an unsupervised learning method.

4 Random Forests for Unsupervised Learning

In the unsupervised learning method [15], which is described in what follows, the training data $\mathcal{D}_{\text{u}}$ consists of a set of input vectors $\boldsymbol{x}_m$ without any corresponding target values. Furthermore, no assumptions are made about the number of clusters potentially present. The goal is to discover groups of similar examples within the data, which is called *clustering*. Intuitively, a cluster represents a group of datapoints whose distances are small compared with the distances to points outside of the cluster [4]. Hence, clustering aims to partition the dataset by finding K clusters $\{C_1, \ldots, C_K\}$ of unlabeled datapoints.

A general approach for the clustering process is depicted in Fig. 3. The dataset is often first subjected to pre-processing. An essential part of this pre-processing is the extraction of relevant features of $\boldsymbol{x}_m$. In the next step, all datapoints $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M_{\text{u}}}\}$

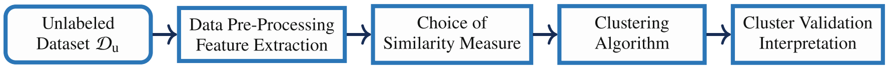
Unlabeled Dataset $\mathcal{D}_\mathrm{u}$
Data Pre-Processing Feature Extraction
Choice of Similarity Measure
Clustering Algorithm
Cluster Validation Interpretation

Fig. 3 A general approach for clustering an unlabeled dataset

of $\mathcal{D}_\mathrm{u}$ are being compared with a similarity measure. This relation is described in terms of the proximity, which can be defined through the similarity or dissimilarity. The proximity measure strongly influences how the data is going to be clustered. Since the proximity between all datapoints of the dataset $\mathcal{D}_\mathrm{u}$ has to be specified, the proximity is a $M_\mathrm{u} \times M_\mathrm{u}$ symmetric matrix $\boldsymbol{P} = (P_{ij})$. The higher the value of P_{ij}, the more similar the datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are. The diagonal elements P_{ii} always have to be one. After performing the similarity comparison between all datapoints, the matrix $\boldsymbol{P}$ is fed into a clustering algorithm, with the aim to partition the dataset into homogeneous groups and to identify patterns. Finally, the clustering results have to be validated and interpreted.

Given the general overview of a clustering procedure, in the following we introduce an unsupervised learning approach based on Random Forests and focus on defining a similarity measure and the choice of the clustering algorithm. A big advantage over many other clustering techniques is, that the proposed method does not require a pre-defined number of clusters.

4.1 Similarity Measure Based on Random Forests

In order to perform unsupervised learning with Random Forests, first a similarity measure is proposed. Therefore, the procedure of growing the trees has to be adapted. In a first step we define a classification task with two classes, A and S, where all datapoints $\boldsymbol{x}_m$ of $\mathcal{D}_\mathrm{u}$ are labeled with A. The tricky part arises with the construction of the second class S. We define S as a synthetic dataset $\mathcal{S} = \{z_1, \ldots, z_K\}, z \in \mathcal{X}$ based on some distribution, such that the synthetic dataset can be considered as noise. Given A and S, the Random Forest is trained just as a normal classification task, described in the previous Sections. The aim here is to distinguish between the given dataset $\mathcal{D}_\mathrm{u}$ and the generated noise dataset $\mathcal{S}$. The underlying mechanism is that, if there is a structure in the data in $\mathcal{D}_\mathrm{u}$ the Random Forest has to fit its leaves to it in order to achieve a low error. Figure 4 illustrates the mechanism of distinguishing between A and S, which leads to the separation of $\mathcal{D}_\mathrm{u}$. Once the Random Forest is trained, two datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ at a time are run through all trees to determine the similarity P_{ij}, similarly as described in Sect. 3.2.2. But, instead of evaluating the proximity solely on leaves, the proposed similarity measure takes into account the full paths of the datapoints through the trees instead. Thereby the complete information provided by the Random Forest is captured, which makes the process robust. The principle behind this path proximity

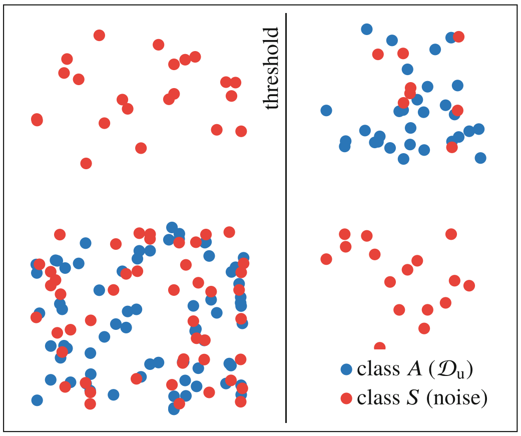

Fig. 4 Assuming a separable structure in the dataset $\mathcal{D}_\mathrm{u}$, the separation between class A and class S will implicitly learn the underlying structure of $\mathcal{D}_\mathrm{u}$

is detailed in Sect. 4.1.2. Before describing the path proximity, we will first examine what needs to be considered when constructing the noise dataset $\mathcal{S}$.

4.1.1 Constructing the Noise Dataset

In [7] the construction of $\mathcal{S}$ is realized by "independent sampling from the one-dimensional marginal distributions of the original data" $\mathcal{D}_\mathrm{u}$, as depicted in Fig. 4. Another construction principle is explained in [21], where $\mathcal{S}$ is build by sampling randomly from an assumed uniform distribution within the Q-dimensional hypercube defined by the minimum and maximum values of $\mathcal{D}_\mathrm{u}$. Independent of the way how the synthetic data is generated, the appearing task is to distinguish between noise and the actual data. One main drawback of the proposed noise generation methods occur with high dimensional input spaces. In order to force the Random Forest to fit properly to the inherent structure, the noise distribution has to be very dense. If there is no more noise data left to perform the dividing task, the resulting leaves become large. Due to the curse of dimensionality [4], for high dimensional input spaces this leads to a huge amount of necessary synthetic datapoints and accordingly to a highly unbalanced ratio between A and S. Even though the sampling from marginal distributions addresses this issue by sampling mainly in the regions of interest, the results in high dimensional spaces might not be satisfactory.

A solution to this problem is to not explicitly generate the noise, but to estimate the required number of noise datapoints in each split. The number of noise points in a node of the tree is chosen to be equal to the number of datapoints from $\mathcal{D}_\mathrm{u}$ in that node. When growing the trees, a randomly chosen distribution from a set of predefined distributions is used in order to construct the noise dataset at each split.

We re-formulate the estimated Gini impurity for the node t in an arbitrary tree as

$$r_g(\mathrm{t}) = \sum_{c=1}^{2} \frac{M_c(\mathrm{t})}{M(\mathrm{t})} \left(1 - \frac{M_c(\mathrm{t})}{M(\mathrm{t})}\right), \tag{45}$$

where $M_c(\mathrm{t})$ the number of datapoints in node t, which belong to class c. The gini gain resulting by splitting t into the child nodes $\mathrm{t_L}$ (left) and $\mathrm{t_R}$ (right) is then

$$\Delta i(\mathrm{t}, \mathrm{t_L}, \mathrm{t_R}) = r_g(\mathrm{t}) - \frac{M(\mathrm{t_L})}{M(t)} r_g(\mathrm{t_L}) - \frac{M(\mathrm{t_R})}{M(\mathrm{t})} r_g(\mathrm{t_R}). \tag{46}$$

The optimal split is given if $\Delta i(\mathrm{t}, \mathrm{t_L}, \mathrm{t_R})$ is maximal. Hence, the number of datapoints of each class in each node (t, $\mathrm{t_L}$ and $\mathrm{t_R}$) is required. The number of original datapoints $M_{\mathcal{D}_{\mathrm{u},b}}\left(\mathrm{t}_{j,b}\right)$ of the bagged dataset $\mathcal{D}_{\mathrm{u},b}$ belonging to the b-th tree in the j-th node $\mathrm{t}_{j,b}$ of this tree, as well as in the possible child nodes can simply be counted. The number of noise datapoints in the same node needs to be estimated for a given split value $\tau_{\tilde{n}}$. Let $z_{\tilde{n}}$ be the standardized value of $\tau_{\tilde{n}}$ (see Eq. (49)). Then, the number of noise datapoints for the left and right child nodes of a node $\mathrm{t}_{j,b}$ is calculated as

$$M_{S,\mathrm{l}_{j,b}}(z_{\tilde{n}}) = M_{\mathcal{D}_{\mathrm{u},b}}\left(\mathrm{t}_{j,b}\right) \mathrm{P}\left(Z_{\tilde{n}} \leq z_{\tilde{n}}\right) \text{ and} \tag{47}$$

$$M_{S,\mathrm{r}_{j,b}}(z_{\tilde{n}}) = M_{\mathcal{D}_{\mathrm{u},b}}\left(\mathrm{t}_{j,b}\right) - M_{S,\mathrm{l}_{j,b}}(z_{\tilde{n}}), \tag{48}$$

where $\tilde{n}$ stands for the $\tilde{n}$-th dimension of the vector whose features are chosen randomly in each node when constructing the trees. The values $M_{S,\mathrm{l}_{j,b}}$ and $M_{S,\mathrm{r}_{j,b}}$ denote the number of corresponding noise points in the left and right child note of $\mathrm{t}_{j,b}$, given that the split $\tau_{\tilde{n}}$ is chosen, since $z_{\tilde{n}}$ is the standardized value of $\tau_{\tilde{n}}$. $\mathrm{P}\left(Z_{\tilde{n}} \leq z_{\tilde{n}}\right)$ is the value of the cumulative density function (cdf) at the standardized threshold $z_{\tilde{n}}$. The standardized threshold $z_{\tilde{n}}$ in the $\tilde{n}$-th dimension is determined with

$$z_{\tilde{n}} = \frac{\tau_{\tilde{n}} - \mu_{\tilde{n}}}{\sigma_{\tilde{n}}}, \tag{49}$$

$$\mu_{\tilde{n}} = \frac{\max\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}} + \min\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}}}{2}, \tag{50}$$

$$\sigma_{\tilde{n}} = \frac{\max\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}} - \min\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}}}{6}, \tag{51}$$

where $\max\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}}$ and $\min\left\{\mathfrak{X}_{\mathrm{t}_{j,b}}\right\}_{\tilde{n}}$ yield the maximum or minimum value of the subspace $\mathfrak{X}_{\mathrm{t}_{j,b}}$ in the dimension specified by $\tilde{n}$. The interval of a node covers $\pm 3\,\sigma_{\tilde{n}}$.

Next, we define a set of distributions. The first distribution used is the uniform distribution, where its cdf is given by

$$\mathrm{P_u}\left(Z_{\tilde{n}} \leq z_{\tilde{n}}\right) = \frac{1}{6} z_{\tilde{n}} + \frac{1}{2}. \tag{52}$$

The standard normal distribution is the second used distribution and is approximated through [22]

$$\mathrm{P_n}\left(\mathsf{z}_{\tilde{n}} \leq z_{\tilde{n}}\right) = \frac{1}{1 + e^{-\sqrt{\pi}\left(\beta_1 z_{\tilde{n}}^5 + \beta_2 z_{\tilde{n}}^3 + \beta_3 z_{\tilde{n}}\right)}}, \tag{53}$$

where $\beta_1 = -0.0004406$, $\beta_2 = 0.04181198$ and $\beta_3 = 0.9$ holds. Third, a bimodal distribution is used, which is build as the sum of two shifted standard normal distributions $\mathrm{P_n}$ with

$$\mathrm{P_b}\left(\mathsf{z}_{\tilde{n}} \leq z_{\tilde{n}}\right) = \mathrm{P_n}\left(\mathsf{z}_{\tilde{n}} - 3 \leq z_{\tilde{n}} - 3\right) + \mathrm{P_n}\left(\mathsf{z}_{\tilde{n}} + 3 \leq z_{\tilde{n}} + 3\right). \tag{54}$$

The randomly selected noise distributions at each split relax the dependency of the proximity measure to one specific distribution. Obviously, the set of three proposed distributions can also be extended with or replaced by other distributions.

Up to this point, we know how to grow the forest and how to compute the noise data to solve the classification task. The last missing element to obtain the data adaptive similarity measure is to describe the proposed path proximity.

4.1.2 Path Proximity

The proposed proximity measure takes into account the full paths of the datapoints through the trees instead of just using the terminal nodes.

Let a Random Forest consist of B trees $\mathfrak{T}$, where the b-th tree $\mathfrak{T}_b$ is constructed based on the bagged dataset $\mathcal{D}_{\mathrm{u},b}$. Then a tree $\mathfrak{T}_b$ consists of N_b nodes $\mathrm{t}_{n,b}$. A path of a datapoint through a tree can be defined by a set including all nodes the datapoint passed. This leads to the path formulation

$$\mathcal{T}_{i,b} = \left\{\mathrm{t}_{1,b}, \mathrm{t}_{n_{i_2},b}, \ldots, \mathrm{t}_{N_i,b}\right\}, \tag{55}$$

where the index i represents the i-th datapoint $\boldsymbol{x}_i$. The node $\mathrm{t}_{1,b}$ is the root node of the b-th tree $\mathfrak{T}_b$ and hence the first node on the path of the i-th datapoint. The node $\mathrm{t}_{n_{i_2},b}$ is the second node on the path, where $n_{i_2,b}$ represents the node number n the datapoint has passed. The last node on the path of the i-th datapoint in the b-th tree is $\mathrm{t}_{N_i,b}$.

In order to compare the paths of two datapoints through the b-th tree, the corresponding sets $\mathcal{T}_{i,b}$ and $\mathcal{T}_{j,b}$ need to be compared. The Jaccard Index [13]

$$P_{ij}(b) = \frac{|\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|}{|\mathcal{T}_{i,b} \cup \mathcal{T}_{j,b}|} \tag{56}$$

$$= \frac{|\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|}{|\mathcal{T}_{i,b}| + |\mathcal{T}_{j,b}| - |\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|} \tag{57}$$

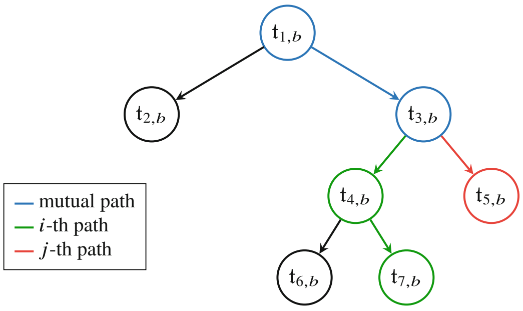

Fig. 5 Path proximity example

is used for this purpose. It holds that $P_{ij}(b) \in (0, 1]$, since at least the root node is present in both sets. This way, a similarity measure, given the two datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, based on the b-th tree is defined. In Fig. 5 an example tree with two arbitrary paths is shown. The mutual path of both datapoints is colored in blue, and the single paths are depicted in green and red. Interpreting Eq. (57) based on Fig. 5 leads to $|\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|$ being the length of the mutual path and $|\mathcal{T}_{i,b}|$ being the length of the i-th path (i-th datapoint) starting from the root node, $|\mathcal{T}_{j,b}|$ respectively. For the depicted example, the corresponding Jaccard index would be $2/5$. In other words, the i-th and j-th datapoints have a similarity of 0.4. A value of 1 indicates, that both datapoints are identical or very similar according to the given tree.

By averaging over all B values P_{ij} (b) in a forest, we obtain the path proximity

$$P_{ij} = \frac{1}{B} \sum_{b=1}^{B} \frac{|\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|}{|\mathcal{T}_{i,b}| + |\mathcal{T}_{j,b}| - |\mathcal{T}_{i,b} \cap \mathcal{T}_{j,b}|}. \tag{58}$$

If two datapoints have the same paths in all trees the proximity will be one. Contrary, if they only share the root nodes, the proximity will be very small tending towards zero. The path proximity enables one to cover more than just the leaf information of the forest within one scalar value. Extracting the Random Forest based path proximity for a given dataset we obtain a data adaptive similarity measure.

With the proximities between all datapoints structured in the similarty matrix $\boldsymbol{P}$, we can advance to the next step according to Fig. 3 by applying a clustering algorithm, which has the task to group similar datapoints given $\boldsymbol{P}$. Clustering algorithms can be categorized into several types and various algorithms, in the following we briefly discuss the *hierarchical clustering* method applied for the proposed unsupervised learning technique with Random Forests.

4.2 Hierarchical Clustering

Hierarchical clustering methods can be distinguished between *agglomerative* or *divisive*. Due to the computation complexity, divisive methods are not commonly used in practice [23]. At the beginning of the clustering process, agglomerative algorithms consider each datapoint as a single cluster, so that the number of datapoints equals the number of clusters. Then, in each iteration a pair of clusters is successively merged until one single cluster remains. This hierarchy can be visualized with a dendrogram. The methods differ in how the dissimilarity from a merged cluster to all the remaining is computed, the so-called *linkage* function. Among several other functions [17], two commonly used linkage functions are the single and average linkage.

For single linkage, the minimum dissimilarity between all the elements of both clusters is used as dissimilarity of the clusters

$$d_{kl} = \min_{\substack{i \in C_k \\ j \in C_l}} \left\{ d_{ij} \right\}, \tag{59}$$

where d_{kl} denotes the dissimilarity between two arbitrary clusters C_k and C_l.

For average linkage, dissimilarity is determined through the average of all dissimilarities between the points of the two clusters

$$d_{kl} = \frac{1}{|C_k||C_l|} \sum_{i \in C_k} \sum_{j \in C_l} d_{ij}, \tag{60}$$

where $|C_k|$ denotes the number of objects in cluster C_k, and $|C_k|$ the number in C_k.

The agglomerative hierarchical clustering results in a hierarchy, the hierarchy can be visualized as dendrogram. When using the order of the leaves in the dendrogram, permutations on $\boldsymbol{P}$ can be performed, such that a reordered proximity matrix ($\boldsymbol{P}_\text{o}$), is obtained. The matrix $\boldsymbol{P}_\text{o}$ represents the clusters in the data and provides a graphical interpretation of the data inherent structure.

4.3 Cluster Analysis and Visualization

Cluster analysis can be performed with a visualization as depicted in Fig. 6. On the right-hand side, the two-dimensional toy dataset, consisting of four clusters with different shapes and densities, is depicted. The proximity matrices are represented as squared images (a)–(d), where dark pixels represent zero entries ($P_{ij} = 0$) and bright pixels with higher similarity. The bright squares along the diagonal represent the four clusters. Squares, which are not aligned the diagonal represent the inter-cluster similarity. It should be noted, that these matrices represent the similarities before applying hierarchichal clustering. The main purpose of Fig. 6 is to show the beneficial effects of the path proximity and ensemble noise.

For example, the compact cluster no. 4 reveals a brighter square on the similarity matrix at the bottom right side along the diagonal axis compared to the widely spread cluster no. 2. That is, because the datapoints within cluster no. 4 share a

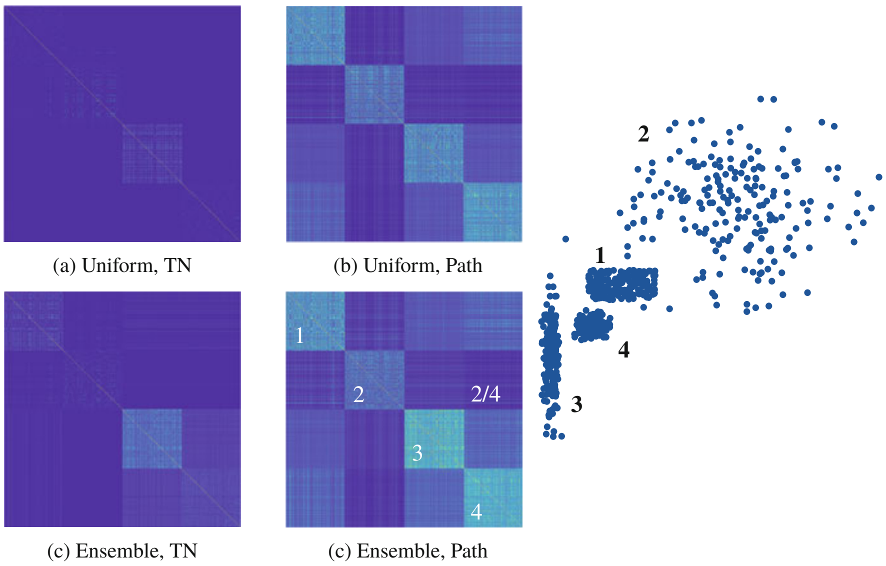

Fig. 6 Comparison of uniformly distributed noise versus ensemble noise, as well as path proximity versus terminal node (TN) for computing the similarity. The right hand-side depicts the two-dimensional toy dataset, consisting of four clusters with different shapes and densities. **(a)** Uniform, TN. **(b)** Uniform, Path. **(c)** Ensemple, TN. **(d)** Ensemple, Path

higher similarity in along both axes compared to cluster no. 2. Additionally, the inter-cluster similarity between both clusters, depicted with '2/4', is low, since the datapoints of the two clusters show almost no overlap along the axes.

In addition, the four subfigures (a)–(d) depict the effects of the noise ensemble compared to a single uniform distribution, as well as extracting the similarity out of the datapoints' paths through the trees instead of only taking the terminal nodes (TN) into account. The similarity of the four clusters can best be identified in subfigure (d), where the ensemble noise and especially the path proximity support an improved similarity measure.

5 Applications

Possible applications of Random Forests in the automotive domain are manifold. In the following, two methods for categorizing traffic scenarios are presented. The first case applies the unsupervised learning method from the previous section for the identification of similar traffic scenarios. The second case demonstrates how Random Forests can be integrated into Deep Learning architectures to tackle the problem of Open-Set recognition for traffic scenarios.

5.1 Traffic Scenario Clustering

Traffic scenario categorization is an important component for downstream tasks like trajectory planning, emergency braking and other functions for autonomous driving. Road traffic does not evolve completely random, since it's framed by the infrastructure, traffic rules, etc. [12], so that traffic scenarios do follow certain patterns, and can be categorized according to the set of features selected. In this section the unsupervised method, as presented in the previous section, is applied to real world traffic scenarios to identify such patterns. Figure 7 depicts the overview of the complete framework, starting from data generation and feature extraction, up to the cluster identification and validation.

First, a set of vehicle trajectories from a public roundabout is extracted with drone imagery by applying the method published in [16, 19]. A second trajectory dataset is recorded on a vehicle test track in order to create scenarios with critical driving maneuvers. The criticality is achieved via strong braking and cut-in maneuvers with small gaps between the vehicles, which are not commonly seen in public traffic. Since the two datasets are recorded at different places, a coordinate transformation is applied as well, so that both can be overlayed on a road map. All scenarios involve at least two vehicles and the timespan is set to 5 s. In total 110 critical scenarios are generated, the same quantity of scenarios of the public road is randomly selected. The extracted trajectories are then geo-referenced and coupled with a road map, which allows one to generate road-adaptive features. Especially for road sections with curvatures and crossings, one has to align the paths driven by vehicles in accordance to the road layout, in order to compute features such as time gaps. The aim of this demonstration is to identify several scenario categories and especially to distinguish between critical and non-critical scenarios.

Given the trajectories and the road information, a set of four features

$$\boldsymbol{x} = [\bar{v}, v_\mathrm{x}, a_\mathrm{x}, \varrho] \tag{61}$$

is extracted from the two datasets. All features relate to the ego vehicle within that 5 s scenario length. $\bar{v}$ denotes the average speed, v_x and a_x denote the minimum

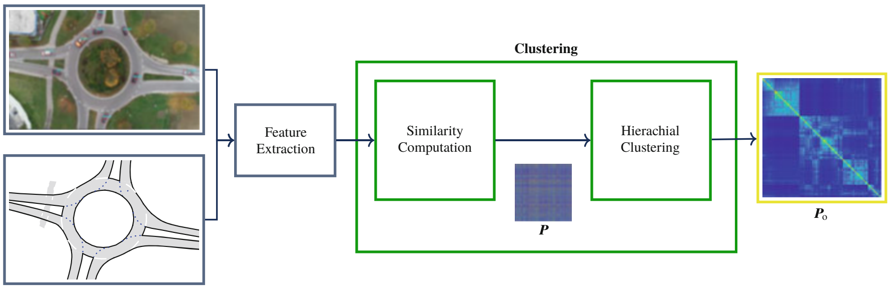

Fig. 7 Traffic scenario clustering: from data generation up to cluster visualization

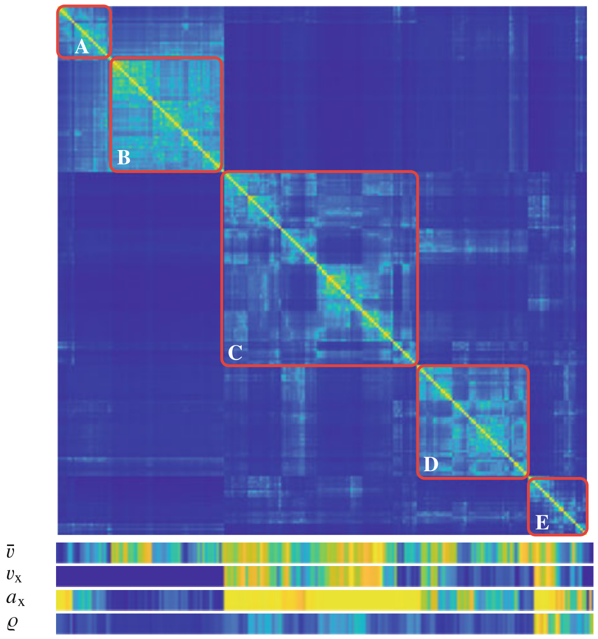

Fig. 8 Proximity matrix: The scenarios are coarsely divided into five clusters A to E. Below, the normalized values for all variables are depicted, where bright coloring indicates high values

longitudinal velocity and acceleration. ϱ denotes the minimum time-headway between the ego and a target vehicle in a crossing scenario, assuming constant velocity, similarly as with the typical time-headway estimation for car-following scenarios. For demonstration purposes the number of features is limited to four, which should separate the critical from the non-critical ones. In general, the choice and number of features has to be aligned according to the application.

As depicted in Fig. 8, five clusters, A to E, are selected. One can recognize smaller clusters within these clusters and differentiate them more fine-grained accordingly. Instead of visually selecting the clusters, one could also use the *elbow* method instead. The clustering result can be physically validated by illustrating the feature values below the proximity matrix, as depicted in Fig. 8. For each cluster one typical scenario is depicted in Fig. 9. The left column in Fig. 9 shows the paths of the vehicles, the two right hand-side columns depict the start and the end of the scenario.

Cluster A and B represent critical crossing scenarios with two vehicles, one gray and one black vehicle. In both cases, the merging vehicle (gray) approaches the roundabout without considering the second vehicle (black). In cluster A, both vehicles are able to brake just before a potential collision. Whereas in cluster B, the merging vehicle continues its drive and violates the right-of-way, hence the black vehicle has to perform a braking. Similarly to B, in cluster D the merging vehicle violates the right-of-way, thus forcing a third vehicle (green) to react and brake.

Cluster C and E contain most of the casual driving scenarios, which were filmed on the public road. Since a few scenarios are randomly selected from a large dataset,

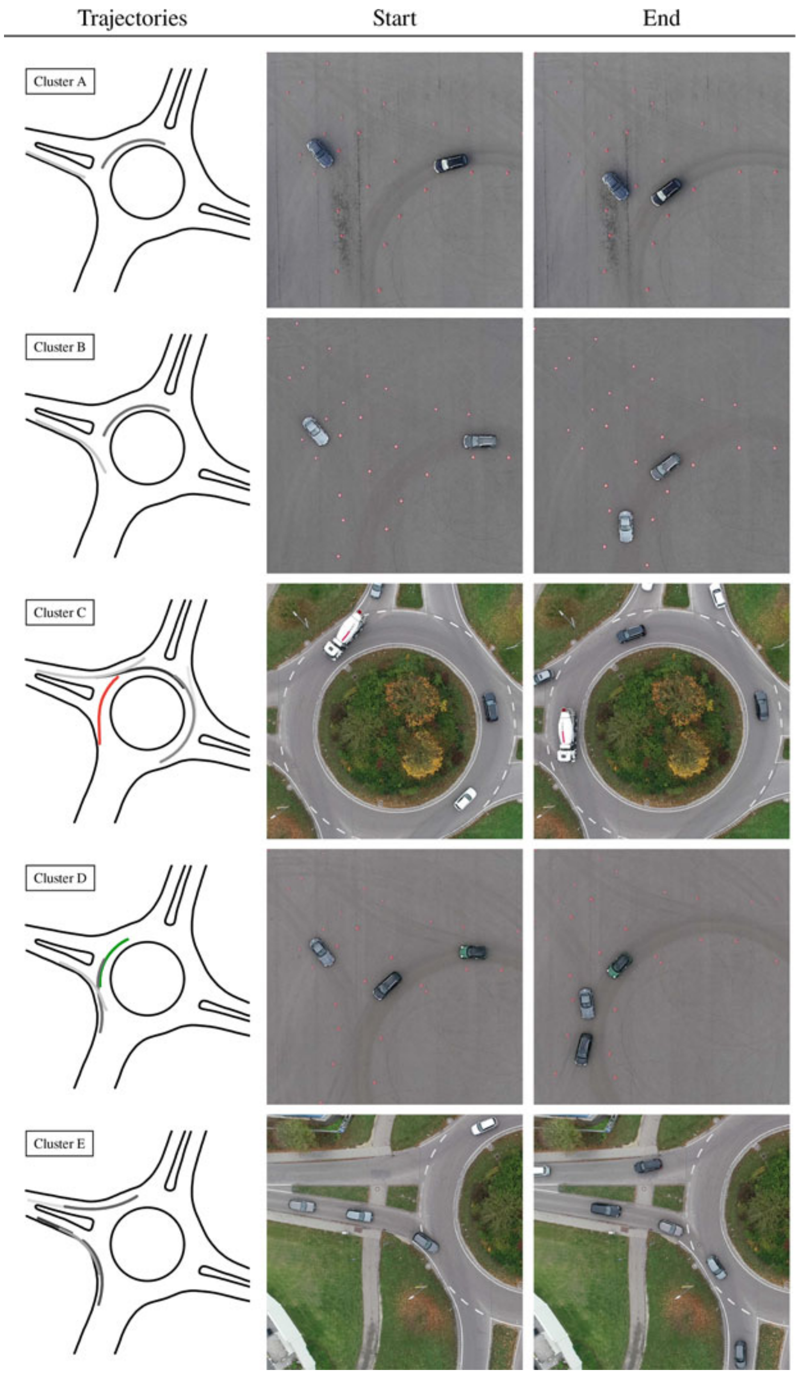

Fig. 9 Typical scenarios from each cluster: Cluster A,B and D represent critical scenarios, cluster C and E casual driving on a public road

these scenarios are correspondingly diverse. The selection and number of features only allows a rough separation. This can be confirmed with the proximity matrix in Fig. 8, where especially for cluster C several smaller clusters can be detected. Cluster C and E contain casual driving scenarios, such as car-following and leaving the roundabout, see the white car depicted in the example for cluster C. The example in cluster E depicts the gray colored ego vehicle approaching the roundabout behind another vehicle and entering the roundabout, while other cars are leaving the scene above.

After the cluster validation, these five groups can be defined as classes. If further data is collected during operation, the new scenarios can be assigned to already known classes. However, one must expect to find novel scenario types. The next section shows how the Random Forest, embedded in a deep learning architecture, can help to deal with new scenarios that cannot be assigned to any of the known classes.

5.2 *Open Set Recognition for Traffic Scenarios*

Typically, learning models proposed in the literatures [9, 18] work under the *closed-world* assumption, which means that the model will classify all the inputs only to one of the K classes used in the training. This is an issue in the real-world, as there are possibilities to encounter new scenario classes when the vehicle is driving on-road. The models trained with closed-world assumptions will fail in the cases where they encounter new classes as the models classify the inputs to only one of the K trained classes. This is a challenging and important problem to be addressed and leads to a new paradigm called *Open-Set Recognition* (OSR) [20].

An OSR model trained on K classes should be able to classify a given input to one of the K classes - or as an unknown. According to [2, 3], simply thresholding a closed-world model with a user defined threshold might not be satisfactory and the performance of such models deteriorates in an open-world case.

OSR models are either distance based, reconstruction based or extreme value based. In [1], a method based on a combination of *Convolutional Neural Networks* (CNN) and a Random Forest is proposed. An overview of the architecture is shown in Fig. 10. The scenarios are represented as a sequence of occupancy grids, with each occupancy grid representing the occupancy of objects and infrastructure in the scene at a time stamp. These grids are fed into the CNN to extract the features. Finally, the classification is done by the Random Forest algorithm combined with extreme value distributions. During the training phase, firstly a CNN is trained on a set of K labelled classes. As a second step, the fully connected layer of the CNN is removed and the flattened output is used as input for a Random Forest algorithm. The Random Forest is trained on a set of extracted features from the CNN for a given training set. In this second phase, the trained Random Forest and the CNN classify an input based on the majority voting scheme by the Random Forest algorithm. In the third step the class-specific vote patterns are modelled using Extreme Value Theory (EVT) based

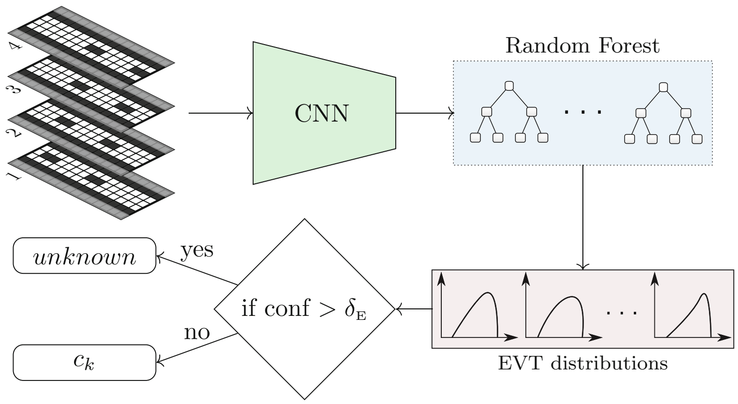

Fig. 10 Open-Set Recognition architecture: CNN for feature extraction, Random Forest and EVT based voting patterns for classification, Figure adapted from [1]

distributions. Class-specific vote patterns collected for the training dataset provide comprehensive information about the uncertainty of the classifier for each class. In the inference or the test phase, the class-specific EVT distributions are used to estimate the probability that a sample belongs to a class c_k or an unknown class based on the number of trees in the Random Forest voting for each class.

The method was tested on real world datasets. The ensemble nature of the Random Forest algorithm when combined with EVT distributions is shown to provide a much more robust OSR accuracy when compared to using other OSR methods and using standard scores like *Softmax* or majority voting.

6 Conclusion

This chapter proposes an unsupervised learning method in order to categorize traffic scenarios. The knowledge about traffic scenario categories is an important aspect for an efficient validation process for automated driving functions. Hence, the scenario categorization has the potential to accelerate the validation process by selecting representatives of each cluster and thereby reducing redundancies by avoiding to test very similar scenarios.

The proposed method is based on Random Forests and performs the pattern recognition only given the input data, i. e., where the availability of labels for training is absent. The goal is to discover groups of similar examples within the data. To achieve this, one has to memorize, compress and structure the data. This can be done by representing a traffic situation with a set of relevant features. These features can then be used to train the proposed method.

The core of the presented method lies in the data-adaptive similarity measure, so that data points can be compared in order to decide, whether they are similar and belong to the same cluster. The underlying mechanism is that, if a separable structure in the dataset is existent, the separation between the actual data and the second, synthetic data, will implicitly learn the underlying structure of the unlabeled dataset. Once the Random Forest is trained, two data points at a time are run through all trees to determine the similarity between both. The similarities between all traffic scenarios can be written in a similarity matrix. By applying hierarchical clustering techniques on that matrix, clusters of traffic scenarios with similar characteristics emerge, while being separated from those with low similarity. The chapter concludes with an exemplified application. It is shown how scenarios, represented by vehicle trajectories, can be categorized according to the vehicle dynamics, as well as the interaction between the traffic participants.

References

1. Balasubramanian, L., Kruber, F., Botsch, M., Deng, K.: Open-set recognition based on the combination of deep learning and ensemble method for detecting unknown traffic scenarios. In: 2021 IEEE Intelligent Vehicles Symposium (IV), pp. 674–681. IEEE (2021)
2. Bendale, A., Boult, T.E.: Towards open world recognition (2014). CoRR abs/1412.5687. https://doi.org/10.1109/cvpr.2015.7298799. http://arxiv.org/abs/1412.5687
3. Bendale, A., Boult, T.E.: Towards open set deep networks (2015). CoRR abs/1511.06233. https://doi.org/10.1109/cvpr.2016.173. http://arxiv.org/abs/1511.06233
4. Bishop, C.M.: Pattern Recognition and Machine Learning, vol. 4 (2006). https://doi.org/10.1117/1.2819119
5. Breiman, L.: Out-of-bag estimation, Technical report, Statistics Department, University of California Berkeley, Berkeley CA 94708 (1996)
6. Breiman, L.: Random forests. Mach. Learn. **45**(1) (2001)
7. Breiman, L.: Using random forests v3.0. Technical Report (2002)
8. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman and Hall/CRC (1984)
9. Cara, I., Gelder, E.D.: Classification for safety-critical car-cyclist scenarios using machine learning. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 1995–2000 (2015). https://doi.org/10.1109/ITSC.2015.323
10. Dietterich, T.G.: Ensemble methods in machine learning. In: Multiple Classifier Systems, LBCS-1857, pp. 1–15. Springer, Berlin (2000)
11. Domingos, P.: A unified bias-variance decomposition. In: Proceedings of 17th International Conference on Machine Learning, pp. 231–238. Morgan Kaufmann Stanford (2000)
12. Gindele, T., Brechtel, S., Dillmann, R.: Learning context sensitive behavior models from observations for predicting traffic situations. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), pp. 1764–1771 (2013). https://doi.org/10.1109/ITSC.2013.6728484
13. Jaccard, P.: The distribution of the flora in the alpine zone.1. New Phytol. **11**(2) (1912)
14. Jain, B.: Condorcet's jury theorem for consensus clustering. In: Trollmann, F., Turhan, A.Y. (eds.) KI 2018: Advances in Artificial Intelligence. Springer International Publishing, New York (2018)
15. Kruber, F., Wurst, J., Morales, E.S., Chakraborty, S., Botsch, M.: Unsupervised and supervised learning with the random forest algorithm for traffic scenario clustering and classification. In: 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 2463–2470 (2019). https://doi.org/10.1109/IVS.2019.8813994

16. Kruber, F., Sánchez Morales, E., Chakraborty, S., Botsch, M.: Vehicle position estimation with aerial imagery from unmanned aerial vehicles. In: IEEE Intelligent Vehicles Symposium (IV) (2020)
17. Murtagh, F., Contreras, P.: Algorithms for hierarchical clustering: an overview. WIREs Data Mining and Knowledge Discovery, pp. 86–97 (2012). https://doi.org/10.1002/widm.53
18. Reichel, M., Botsch, M., Rauschecker, R., Siedersberger, K.H., Maurer, M.: Situation aspect modelling and classification using the scenario based random forest algorithm for convoy merging situations. In: 13th International IEEE Conference on Intelligent Transportation Systems, pp. 360–366 (2010)
19. Sánchez Morales, E., Kruber, F., Botsch, M., Huber, B., García Higuera, A.: Accuracy characterization of the vehicle state estimation from aerial imagery. In: IEEE Intelligent Vehicles Symposium (IV) (2020)
20. Scheirer, W.J., Jain, L.P., Boult, T.E.: Probability models for open set recognition. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2317–2324 (2014). https://doi.org/10.1109/TPAMI.2014.2321392
21. Shi, T., Horvath, S.: Unsupervised learning with random forest predictors. J. Comput. Graph. Stat. **15**(1) (2006). https://doi.org/10.1198/106186006X94072
22. Waissi, G.R., Rossin, D.F.: A sigmoid approximation of the standard normal integral. Appl. Math. Comput. **77**(1) (1996)
23. Xu, R., WunschII, D.: Survey of clustering algorithms. IEEE Trans. Neural Netw. **16**(3) (2005)