Forest GUMP: A Tool for Explanation

Alnis Murtovi(✉), Alexander Bainczyk, and Bernhard Steffen

Chair for Programming Systems, TU Dortmund University, Dortmund, Germany
{alnis.murtovi,alexander.bainczyk,bernhard.steffen}@tu-dortmund.de

Abstract. In this paper, we present Forest GUMP (for Generalized, Unifying Merge Process) a tool for providing tangible experience with three concepts of explanation. Besides the well-known *model explanation* and *outcome explanation*, Forest GUMP also supports *class characterization*, i.e., the precise characterization of all samples with the same classification. Key technology to achieve these results is algebraic aggregation, i.e., the transformation of a Random Forest into a semantically equivalent, concise white-box representation in terms of Algebraic Decision Diagrams (ADDs). The paper sketches the method and illustrates the use of Forest GUMP along an illustrative example taken from the literature. This way readers should acquire an intuition about the tool, and the way how it should be used to increase the understanding not only of the considered dataset, but also of the character of Random Forests and the ADD technology, here enriched to comprise infeasible path elimination.

Keywords: Random Forest, Binary/Algebraic Decision Diagram, Aggregation, Infeasible Paths, Explainability, Random Seed

1 Introduction

Random Forests are one of the most widely known classifiers in machine learning [3,17]. The method is easy to understand and implement, and at the same time achieves impressive classification accuracies in many applications. Compared to other methods, Random Forests are fast to train and they are clearly more suitable for smaller datasets. In contrast to a single decision tree, Random Forests, a collection of many trees, do not overfit as easily on a dataset and their variance decreases with their size. On the other hand, Random Forests are considered black-box models because of their highly parallel nature: following the execution of Random Forests means, in particular, following the execution in all the involved trees. Such black-box executions are hard to explain to a human user even for very small examples.

In contrast, decision trees are considered white-box models because of their sequential evaluation nature. Even if a tree is large in size, a human can easily follow its computation step by step by evaluating (simple) decisions at each node from the root to a leaf. Indeed, the set of decisions along such an execution path precisely explains why a certain choice has been taken.

D. Fisman and G. Rosu (Eds.): TACAS 2022, LNCS 13244, pp. 314–331, 2022.
https://doi.org/10.1007/978-3-030-99527-0_17

Popular methods towards explainability try to establish some user intuition. For example, they may hint at the most influential input data, like highlighting or framing the area of a picture where a face has been identified. Such information is very helpful, and it helps in particular to reveal some of the "popular" drastic mismatches incurred by neural networks: if the framed area of the image does not contain the "tagged" object, the identification is clearly questionable. However, even in a correct classification, the tag by itself gives no reason why the identification is indeed correct.

More ambitious are methods that try to turn black-box model into white-box models, ideally preserving the semantics of the classification function. For Random Forests this has been achieved for the first time in [10,14] using the 'aggregating power' of Algebraic Decision Diagrams (ADDs) and Binary Decision Diagrams (BDDs). ADDs are essentially decision trees whose leaves are labelled with elements of some algebra, whereas BBDs are the special case for the algebra of Boolean values. Lifting the algebraic operations from the leaves to the entire ADDs/BDDs allows one to aggregate entire Random Forests into single semantically equivalent ADDs, the precondition for solving three explainability problems:

- The *Model Explanation Problem* [15], i.e. the problem of making the model as a whole interpretable, is solved in terms of an ADD that specifies precisely the same classification function as the original Random Forest (cf. Section 6.2).
- The *Class Characterization Problem*, i.e. the problem, given a class c, characterizing the set of all samples that are classified by the Random Forest as c. This problem is solved in terms of a BDD which precisely characterizes this set of samples (cf. Section 6.3).
- The *Outcome Explanation Problem* [15], i.e. the problem of explaining a concrete classification, is solved in terms of a minimal conjunction of (negated) decisions that are sufficient to guide the sample into the considered class (cf. Section 6.4).

In this paper, we present Forest GUMP (for Generalized, Unifying Merge Process) a tool for providing a tangible experience with the described concepts of explanation. Experimentation with Forest GUMP does not only yield semantically equivalent, concise white-box representations for a given Random Forest which reveal characteristics of the underlying datasets, but it also allows one to experience, e.g., the impact of random seeds on both the quality of prediction and the size of the explaining models (cf. Section 6). Our implementation relies on the standard Random Forest implementation in Weka [28] and on the ADD implementation of the ADD-Lib [9,12,26]. For a more detailed description of the transformations and a quantitative analysis we refer the reader to [10,11,14].

Related Work: Various methods for making Random Forests interpretable exist such as extracting decision rules from the considered black-box model [6], methods that are agnostic to the black-box model under consideration [20,24] or by deriving a single decision tree from the black-box model [5,7,16,27,29]. In this

context, single decision trees are considered key to a solution of both, the model explanation and outcome explanation problem. State of the art solutions to derive a single decision tree from a Random Forest are approximative [5,7,16,27,29]. Thus, their derived explanations are not fully faithful to the original semantics of the considered Random Forest. This is in contrast to our ADD-based aggregation, which precisely reflects the semantics of the original Random Forest.

After a short introduction to Random Forests in Section 2, we present our approach to their aggregation in Section 3 which is followed by an elimination of redundant predicates from the decision diagrams in Section 4 and a non-compositional abstraction in Section 5. Section 6 introduces Forest GUMP and solutions to the three explainability problems. In the end, we summarize the lessons we have learned using Forest GUMP in Section 7 which is followed by a conclusion and direction to future work in Section 8.

2 Random Forests

Learning Random Forests is a quite popular, and algorithmically relatively simple classification technique that yields good results for many real-world applications. Its decision model generalises a training dataset that holds examples of input data labelled with the desired output, also called *class*. As its name suggests, an ensemble of decision trees constitutes a Random Forest. Each of these trees is itself a classifier that was learned from a random sample of the training dataset. Consequently, all trees are different in structure, they represent different decision functions, and can yield different decisions for the very same input data.

To apply a Random Forest to previously unseen input data, every decision tree is evaluated separately: Tracing the trees from their root down to one of the leaves yields one decision per tree, i.e. the predicted class. The overall decision of the Random Forest is then derived as the most frequently chosen class, an aggregation commonly referred to as *majority vote*. The key advantage of this approach is, compared to single decision trees, the reduced variance. A detailed introduction to Random Forests, decision trees, and their learning procedures can be found in [3,17,23].

In this paper, we use Weka [28] as our reference implementation of Random Forests. However, our approach does not depend on implementation details and can be easily adapted to other implementations.

Figure 1 shows a small Random Forests that was learned from the popular Iris dataset [8]. The dataset lists dimensions of Iris flowers' sepals and petals for three different species. Using this forest to decide the species on the basis of given measurements requires to first evaluate the three trees individually and to subsequently determine the majority vote. This effort clearly grows linearly with the size of the forest. In the following we use this example to illustrate our approach of forest aggregation for explainability.

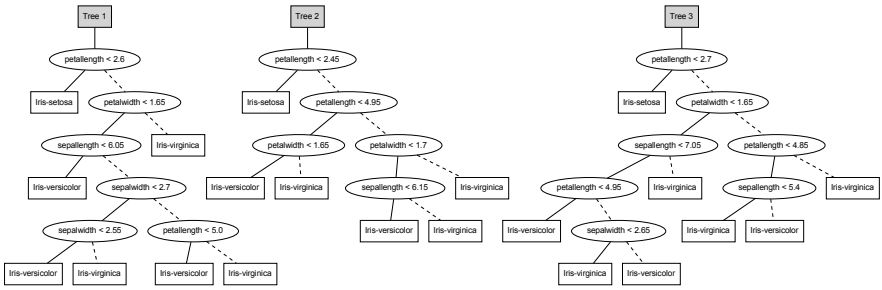

Fig. 1. Random Forest learned from the Iris dataset [8] (39 nodes).

Key idea behind our approach is to partially evaluate the Random Forests at construction time which, in particular, eliminates redundancies between the individual trees of a Random Forest. E.g., in our accompanying Iris flower example (cf. Fig. 1) the predicate $petalwidth < 1.65$ is used in all three trees. This can easily lead to cases where the same predicate is evaluated many times in the classification process. The partial evaluation proposed in this paper transforms Random Forests into decision structures where such redundancies are totally eliminated.

An adequate data structure to achieve this goal for binary decisions are Binary Decision Diagrams [1,4,19] (BDDs): For a given predicate ordering, they constitute a normal form where each predicate is evaluated at most once, and only if required to determine the final outcome.

Algebraic Decision Diagrams (ADDs) [2] generalise BDDs to capture functions of the type $\mathbb{B}^{\mathcal{P}} \to \mathcal{C}^n$ which are exactly what we need to specify the semantics of Random Forests for a classification domain $\mathcal{C}$. Moreover, in analogy to BDDs, which inherit the algebraic structure of their co-domain $\mathbb{B}$, ADDs also inherit the algebraic structure of their co-domains if available.

We exploit this property during the partial evaluation of Random Forests by considering the class vector co-domain (cf. Sect. 3). The aggregation to achieve the corresponding optimised decision structures is then a straightforward consequence of the used ADD technology.

3 Class Vector Aggregation

Class vectors faithfully represent the information about how many trees of the original Random Forest voted for a certain outcome. Obviously, this information is sufficient to obtain the precise results of a corresponding majority vote. Formally, the domain of *class vectors* forms a monoid

$$V := (\mathbb{N}^{|\mathcal{C}|}, +, \mathbf{0})$$

where addition $+$ is defined component-wise and $\mathbf{0}$ is the neutral element.

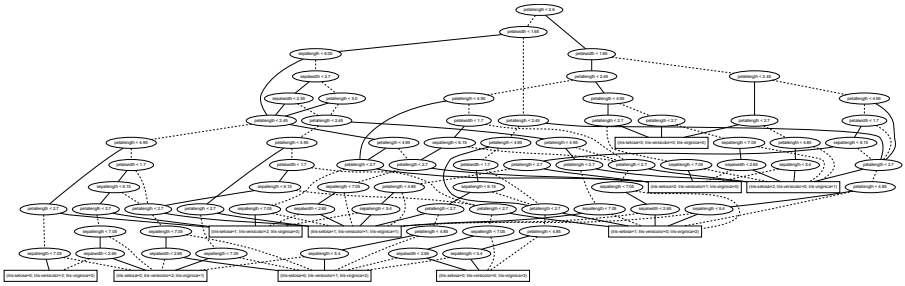

Fig. 2. Class vector aggregation of the Random Forest (83 nodes).

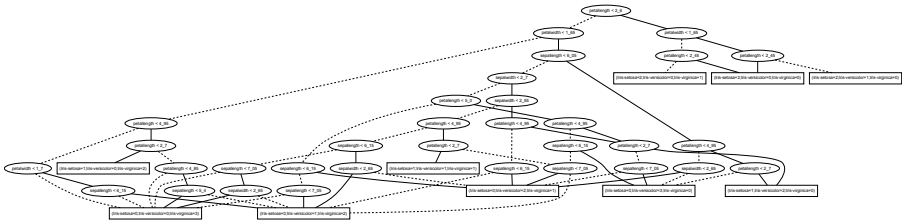

Fig. 3. Class vector aggregation of the Random Forest without semantically redundant nodes (43 nodes).

With the compositionality of the algebraic structure V and the corresponding ADDs $\mathcal{D}_V$, we can transform any Random Forest incrementally into a semantically equivalent ADD. Starting with the empty Random Forest, i.e. the neutral element $\mathbf{0}$, we consider one tree after the other, aggregating a growing sequence of decision trees until the entire forest is entailed in the new decision diagram. The details of this transformation are described in [14]. Figure 2 shows the result of this transformation for our running example.

4 Infeasible Path Elimination

When aggregating the trees of a Random Forest they all use varying sets of predicates. In contrast to simple Boolean variables, predicates are not independent on one another, i.e. the evaluation of one predicate may yield some degree of knowledge about other predicates. E.g., the predicate $petallength < 2.45$ induces knowledge about other predicates that reason about *petallength*: When the petal length is smaller than 2.45 it cannot possibly be greater or equal to 2.7 at the same time. This is not taken care of by the symbolic treatment of predicates we followed until now. In fact, predicates are typically considered independent in the ADD/BDD community.

Infeasible path elimination, as illustrated by the difference between Figure 2 and Figure 3 for our running example, leverages the potential of a semantic

treatment of predicates with significant effect on the size of the resulting ADDs. In fact, the experiments with thousands of trees reported in [14] would not have been successful without infeasible path elimination.

Please note that infeasible path elimination

- is only required after aggregation: The trees in the original Random Forest have no infeasible paths by construction. They are introduced in the course of our *symbolic* aggregation, which is insensitive to semantic properties.
- is compositional and can therefore be applied during the stepwise transformation, before the final most frequent label abstraction (cf. Sect. 5), and at the very end.
- does not support normal forms: Whereas class vector abstraction is canonical for a given variable ordering, infeasible path elimination is not! Thus our approach may yield different decision diagrams depending on the order of tree aggregation. It is guaranteed, however, that the resulting decision diagrams are minimal.

Infeasible path elimination is a hard problem in general.[1] Our corresponding implementation uses SMT-solving [21] to eliminate all infeasible paths. An in-depth discussion of infeasible path elimination is a topic in its own and beyond the scope of this paper.

Class vector aggregation and infeasible path elimination are both compositional and can therefore be applied in arbitrary order without changing the semantics. The majority vote at compile time described in the next section is not compositional and must therefore be applied at the very end.

5 Majority Vote at Compile Time

As mentioned above, maintaining the information about the result of the majority votes is not compositional. In fact, knowing the result of the majority votes for two Random Forest gives no clue about the majority vote of the combined forest. Thus the majority vote abstraction can only be applied at the very end, after the entire aggregation has been computed compositionally.

The result of the compositional aggregation process, including infeasible path elimination, is a decision diagram $d \in \mathcal{D}_V$ with class vectors in its terminal nodes. The majority vote abstraction $\Delta_C : \mathcal{D}_V \to \mathcal{D}_C$ can now be defined as the lifted version of the majority vote abstraction on class vectors $\mathbf{v} \in \mathbb{N}^{|\mathcal{C}|}$ (cf. [14]):

$$\delta_C(\mathbf{v}) := \arg\max_{c \in \mathcal{C}} \mathbf{v}_c.$$

Note that δ_C does not project into the same carrier set but rather from one algebraic structure V into another C. However, these transformations can be applied to the corresponding decision diagrams in the very same way. Fig. 4 shows the result of the most frequent class abstraction for our running example.

[1] For the cases considered here it is polynomial, but there are of course theories for which it becomes exponentially hard or even undecidable.

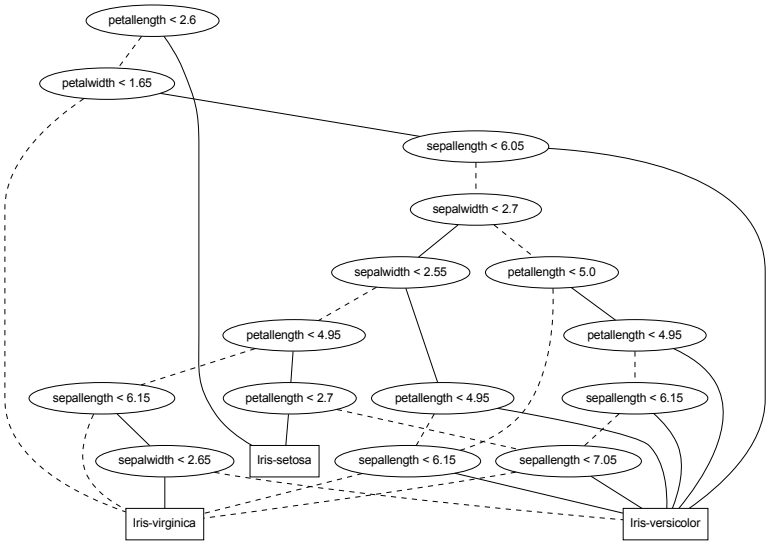

Fig. 4. Most frequent label abstraction of the aggregated Random Forest (majority vote) without semantically redundant nodes (18 nodes).

6 Forest GUMP and Three Problems of Explanability

Forest GUMP[2] (Generalized Unifying Merge Process) is a tool we developed to illustrate the power of algebraic aggregation for the optimization and explanation of Random Forests. It is designed to allow everyone, in particular people without IT or machine learning knowledge, to experience the nature of Random Forests. To avoid unnecessary entry hurdles, we decided to implement Forest GUMP as a simple to use web application. It allows the user to experience the methods described in the previous sections and the proposed solutions to the explanability problems which will be illustrated in the following sections. We will first give a brief overview of Forest GUMP and then showcase its potential in the following sections.

Forest GUMP's user interface (see Figure 5) is essentially divided into two parts. On the left side the user can input the necessary data to learn a Random Forest and subsequently visualize it while the currently chosen representation will be visualized on the right side. First, the user has to upload a dataset or choose one of six datasets that we provide (cf. (1) in Fig. 5) on which the Random Forest will be learned. Next, the hyperparameters necessary for the learning procedure have to be selected, such as the number of trees to be learned (cf. (2) in Fig. 5). Then, one can choose different aggregation methods, i.e. the ones

[2] A link to a running instance of Forest GUMP is available at https://gitlab.com/scce/forest-gump.

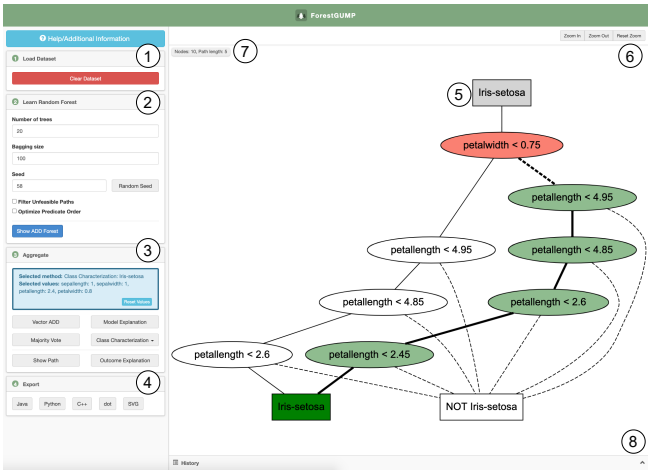

Fig. 5. Overview of Forest GUMP. The visualized ADD is our solution to the class class characterization problem (cf. Sect. 6.3) for the class Iris-Setosa.

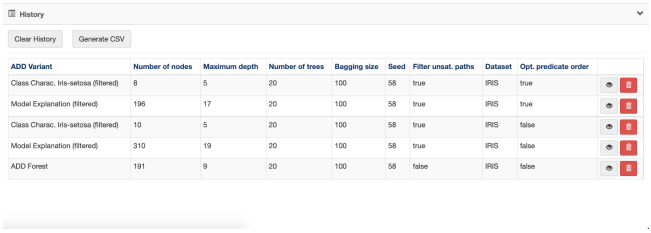

Fig. 6. The execution history in Forest GUMP.

mentioned in the previous sections and further ones which will be explained in the following Sections (cf. (3) in Fig. 5). It it also possible to input a sample, classify it with the ADD and highlight the path from the root the leaf (satisfied predicates are highlighted in green, unsatisfied predicates are highlighted in red). In the end, the currently visualized ADD can be exported as Forest GUMP provides code generators for Java, C++, Python and GraphViz's dot format (cf. (4) in Fig. 5). Additionally, the currently visualized ADD can be exported as an SVG to be viewed locally (cf. (4) in Fig. 5).

The grey rectangle (cf. (6) in Fig. 5) points to the root of the currently visualized ADD. One can zoom into/out which can be helpful when the ADDs are rather large (cf. (6) in Fig. 5). On the top left the number of nodes and the length of the currently highlighted path are displayed (cf. (7) in Fig. 5). On the bottom right, one can open a history of all the representations one chose to visualize (cf. (8) in Fig. 5).

Figure 6 shows the expanded execution history. For each visualized ADD, the execution history lists the aggregation variant, the hyperparameters used to learn

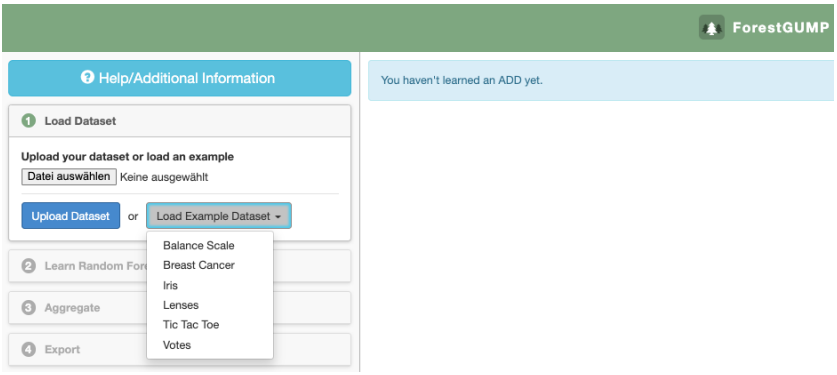

Fig. 7. The user can either choose to upload their own dataset or select one of six exemplary datasets.

the Random Forest and the size (i.e. the number of nodes) and the maximum depth which is the longest path from root to leaf. The execution history also allows one to replay an experiment by clicking on the button on the right side of a row which allows one to compare different ADD variants. One can also delete the individual entries or the whole history and export the history to a CSV.

6.1 A Walkthrough of Forest GUMP

In the following we will see how hard it is to understand how a Random Forest comes to its decision and provide methods for solving the three explainability problems with absolute precision.

Learning a Random Forest To begin, we need a Random Forest which requires a dataset on which it will be learned. In Forest GUMP, the user can upload their own dataset in the Attribute-Relation File Format (ARFF) [28]. Alternatively, we provide six exemplary datasets from which a user can select one to directly start using the tool. Figure 7 illustrates how this looks like in Forest GUMP. Having chosen a dataset, next, the hyperparameters necessary for the learning procedure of the Random Forest have to be specified (see Figure 8). The inputs are the following:

- the *number of trees* to be learned,
- the *bagging size*, i.e. the fraction of samples to be used to learn each tree and
- a *seed* to be able to reproduce the setting.[3]

Additionally, the user can decide to eliminate the infeasible paths as this can strongly reduce the size of the ADDs (see Section 4). While the predicate order is fixed by default, the user can decide to let Forest GUMP optimize the predicate order as the order can also greatly impact the size of the ADDs. A more in

[3] One can generate a random seed by clicking on the button next to the input field.

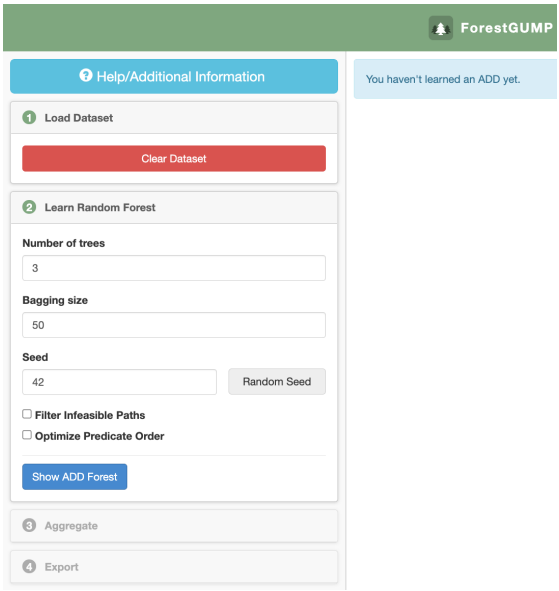

Fig. 8. The user has to specify the necessary hyperparameters to be able to learn a Random Forest. While the first three hyperparameters are needed for the learning procedure, the elimination of the infeasible paths and the optimization of the predicate order are specific to our aggregation method.

depth discussion on the interplay between the infeasible path elimination and the predicate order will follow. Figure 9 shows a Random Forest that was learned on the Iris dataset, consisting of 20 trees[4], a bagging size of 100% and 58 as the seed. If we now want to classify a given input, for each tree we would have to traverse from the root to the leaf and receive one predicted class per tree. The class which was predicted most often is the final result. Trying to understand why the Random Forest predicted this specific class is seemingly impossible. In the following we will show how we can do better.

6.2 Model Explanation Problem

The canonical white-box model corresponding to the Random Forest of Figure 9 can be constructed through the most frequent label abstraction (see Sect. 5) of the aggregated Random Forest (see Sect. 3), whose infeasible paths are eliminated (see Sect. 4). This solves the Model Explanation Problem.

Figure 10 sketches the result of this construction: A canonical white-box model with 310 nodes. Admittedly, this model is still frightening, but given a sample, it allows one to easily follow the corresponding classification process, and in this case it may require at most 19 individual decisions based on the petal

[4] Note that each decision tree is represented as an ADD.

Fig. 9. A Random Forest consisting of 20 individual decision trees (191 number of nodes, longest path consists of 9 nodes). Note that each decision tree is represented as an ADD and that all ADDs share common subfunctions, i.e. it is essentially a shared ADD forest. The actual Random Forest, where nothing is shared, contains 284 nodes.

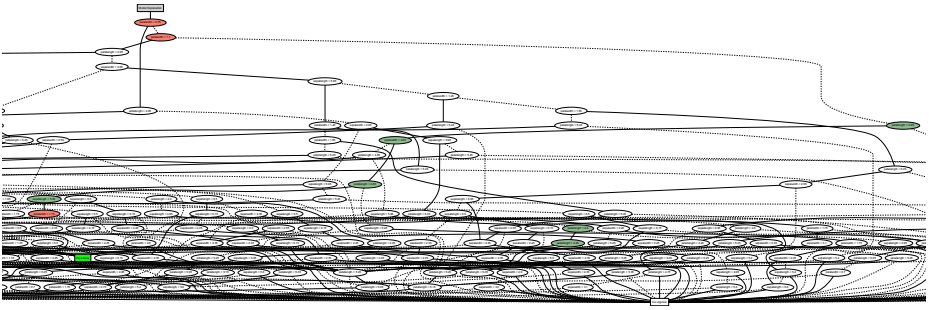

Fig. 10. An extract of the model explanation. The ADD is constructed from the most frequent label abstraction of the aggregated Random Forest following an elimination of all infeasible paths (310 nodes, longest path with length 19, the highlighted path has a length of 9).

and sepal characteristics. This decision set is our set of predicates. The conjunction of these predicates is a solution to the Outcome Explanation Problem. However, more concise explanations are derived from the class characterization BDD discussed in the following section.

Given the sample $petallength = 2.4$, $petalwidth = 1.8$, $sepallength = 5.9$, $sepalwidth = 2.5$, the outcome explanation given by the model explanation consists of the following 9 predicates (in Figure 10 satisfied predicates are highlighted in green, unsatisfied predicates are highlighted in red):

$$\begin{aligned} &\neg(petalwidth < 0.75) \wedge \neg(petalwidth < 1.7) \wedge (petallength < 4.95) \wedge \\ &\quad (sepalwidth < 2.65) \wedge (petallength < 4.85) \wedge (sepallength < 5.95) \wedge \\ &\neg(petalwidth < 1.75) \wedge \quad (petallength < 2.6) \wedge (petallength < 2.45) \end{aligned}$$

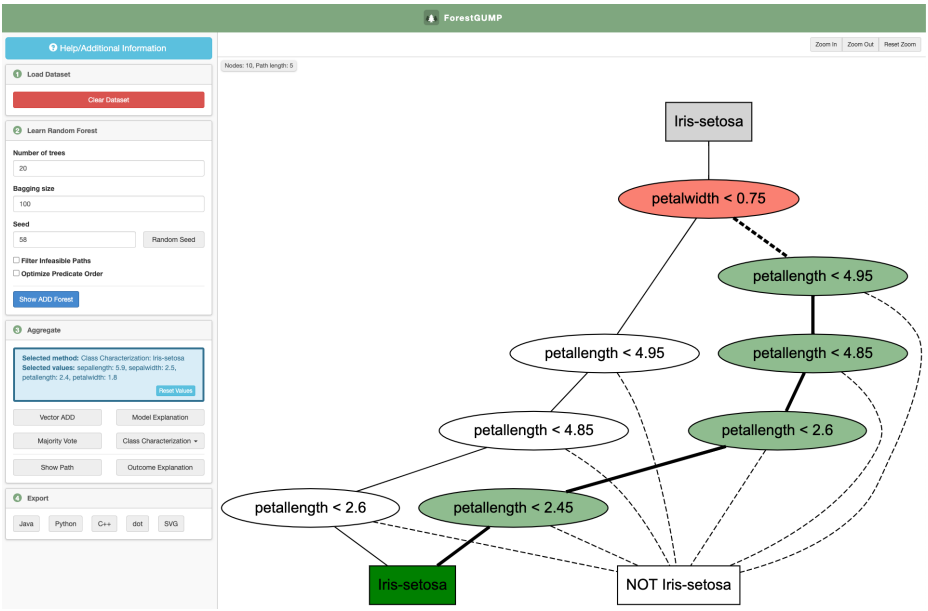

Fig. 11. The class characterization for the class Iris-Setosa (10 nodes, the highlighted path is also the longest path with length 5). The leaf corresponding to Iris-Setosa is highlighted in green, the leaf representing all other classes (i.e. Iris-Virginica and Iris-Versicolor) is highlighted in red.

While this is already an improvement compared to the Random Forest, where you would have to traverse all 20 decision trees, we will see how we can improve even more in the following.

6.3 Class Characterization Problem

The class characterization problem is particularly interesting because it allows one to 'reverse' the classification process. While the direct problem is 'given a sample, provide its classification', the reverse problem sounds 'given a class, what are the characteristics of all the samples belonging to this class?'

BDD-based Class Characterisation can be defined via the following simple transformation function: Given a class $c \in \mathcal{C}$, we define a corresponding projection function $\delta_B(c) : \mathcal{C} \to \mathbb{B}$ on the co-domain as

$$\delta_B(c)(c') := \begin{cases} 1 & \text{if } c' = c \\ 0 & \text{otherwise.} \end{cases}$$

for $c' \in \mathcal{C}$. Again, the function $\delta_B(c)$ can be lifted to operate on ADDs, yielding $\Delta_B(c) : \mathcal{D}_C \to \mathcal{D}_{\mathbb{B}}$.

The BDD shown in Figure 11 is a minimal characterization of the set of all the samples that are guaranteed to be classified as Iris-Setosa.

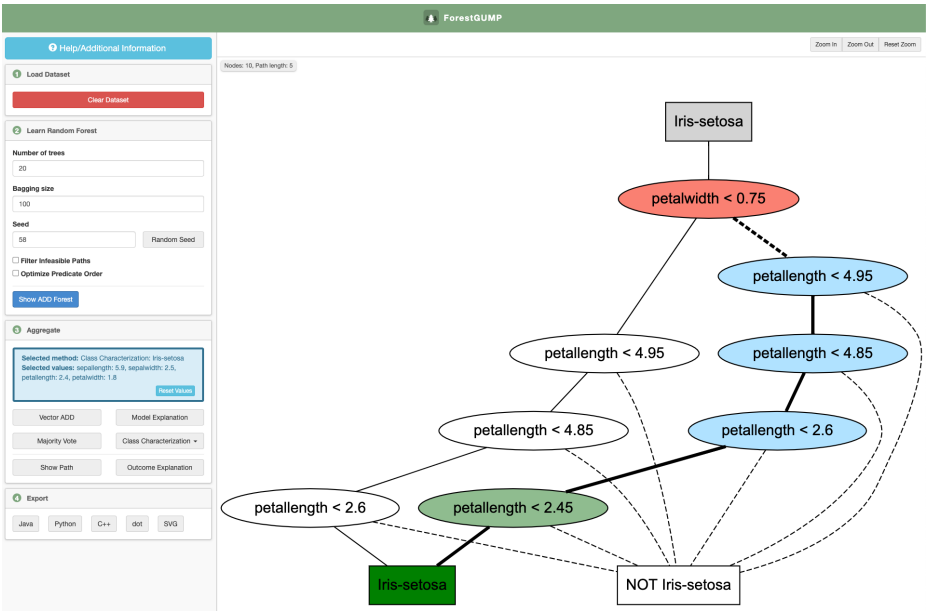

Fig. 12. The outcome explanation for the input $petallength = 2.4$, $petalwidth = 1.8$, $sepallength = 5.9$, $sepalwidth = 2.5$ (10 nodes, hightlighted path of length 5).

Being able to reverse a learned classification function has a major practical importance. Think, e.g., of a marketing research scenario where data have been collected with the aim to propose bestfitting product offers to customers according to their user profile. This scenario can be considered as a classification problem where the offered product plays the role of the class. Now, being able to reverse the customer $\rightarrow$ product classification function provides the marketing team with a tailored product $\rightarrow$ customer promotion process: for a given product, it addresses all customers considered to favor this very product as in the corresponding patent [18].

The path highlighted in Figure 11 is the path from the root to the leaf for the same sample $petallength = 2.4$, $petalwidth = 1.8$, $sepallength = 5.9$, $sepalwidth = 2.5$. Compared to the path with length 9 in the model explanation, we now have a path of length 5 with the following predicates:

$$\neg(petalwidth < 0.75) \wedge (petallength < 4.95) \wedge (petallength < 4.85) \wedge (petallength < 2.6) \wedge (petallength < 2.45)$$

6.4 Outcome Explanation Problem

The previous classification formula expresses the collection of 'conditions' that this sample satisfies, and it provides therefore a precise justification why it is classified in this class. Despite the fact that the class characterization BDD is

canonical, it is easy to see that there are some redundancies in the formula. For example, a $petallength < 2.45$ is also inherently smaller than 2.6, 4.85 and 4.95; therefore, for this specific sample those three predicates are redundant. This is the result of the imposed predicate ordering in BDDs: all the BDD predicates are listed, and they are listed in a fixed order. After eliminating these redundancies, we are left with the following precise minimal outcome explanation: this sample is recognized as belonging to the class Iris-Setosa because it has the properties $\neg(petalwidth < 0.75) \wedge (petallength < 2.45)$.

In Forest GUMP we make these redundant predicates explicit by highlighting them in blue (see Figure 12). From 9 predicates in the model explanation to 5 predicates in the class characterization, we have now arrived at an explanation that only consists of 2 predicates.

7 Lessons Learned

Playing with Forest GUMP led to interesting observations not only concerning the analyzed data domains but also concerning Random Forest Learning and the applied ADD technology.

Random Forest Learning. Changing the random seed for the learning process had a significant impact on the size of the explanation models and the class characterizations. The observed sizes of the explanation models ranged from 138 to 519. Interesting was that the larger sizes did not necessarily imply a better prediction quality. The same also applied to the class characterizations. In fact, we observed a 100% prediction quality for a class characterization of only 3 nodes, while a class characterization for the same species with 40 nodes only scored 33% prediction.

Analyzed Data Domain. The class characterizations for the three iris species differed quite a bit. For two species the observed sizes were much bigger than the sizes of the third species, independently of the chosen random seed and bagging size. In fact, for Iris-Setosa we observed a class characterization with only 3 nodes implying an outcome explanation for our chosen sample with only one predicate. Figure 13 serves for the corresponding explanation. Put it differently, class characterizations seem to be good indications for 'tightness': The closer the samples lie the more criteria are required for separation.

ADD Technology. ADDs are canonical as soon as one has chosen a predicate/variable ordering. Although we could observe the effect of corresponding optimization heuristics[5], the impact was moderate and helpful mainly for model explanation and class characterization. Figure 14 shows the the outcome explanation for the same problem but where the ADD, representing the class characterization for the class Iris-Setosa, is reordered.[6] While the reordering

[5] CUDD [25] provides a number of heuristics for optimizing variable orders.

[6] The used reordering method is named CUDD_REORDER_GROUP_SIFT_CONV as it was both, fast and effective, in our experiments.

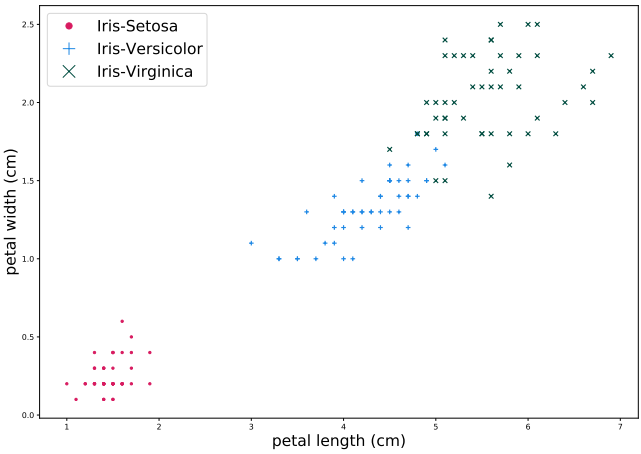

Fig. 13. Visualization of the iris dataset using only the petal length and petal width.

reduces the class characterization size from 10 to 8 nodes, the length of the outcome explanation is unchanged. For the model explanation of Figure 10, the size can be reduced from 310 nodes to 196 nodes while the path for the sample $petallength = 2.4$, $petalwidth = 1.8$, $sepallength = 5.9$, $sepalwidth = 2.5$ actually increased by 1 (from 9 to 10). Thus the outcome explanation may even be impaired. This is not too surprising as these optimizations aim a size reduction and not depth reduction of the considered ADDs. We are currently investigating good heuristics for depth reduction.

More striking was the impact of infeasible path elimination. In fact, this optimization can be regarded key for scalability when increasing the forest size. [14] reports results about forests with 10.000 trees. Without infeasible path reduction already 100 trees are problematic.

Standard ADD frameworks work on Boolean variables rather than predicates. Thus in their setting infeasible paths do not occur. The problem of infeasible path reduction in ADDs was first discussed in [13,14]. Our current corresponding solution is still basic. We are currently generalizing our solution using more involved SMT technology.

Of course, these observations where made on rather small datasets and it has to be seen how well they tranfer to more complex scenarios. We believe, however, that they indicate general phenomena whose essence remains true in larger setting.

8 Conclusion and Perspectives

We have presented Forest GUMP (for Generalized, Unifying Merge Process) a tool for providing tangible experience with three concepts of explanation: *model*

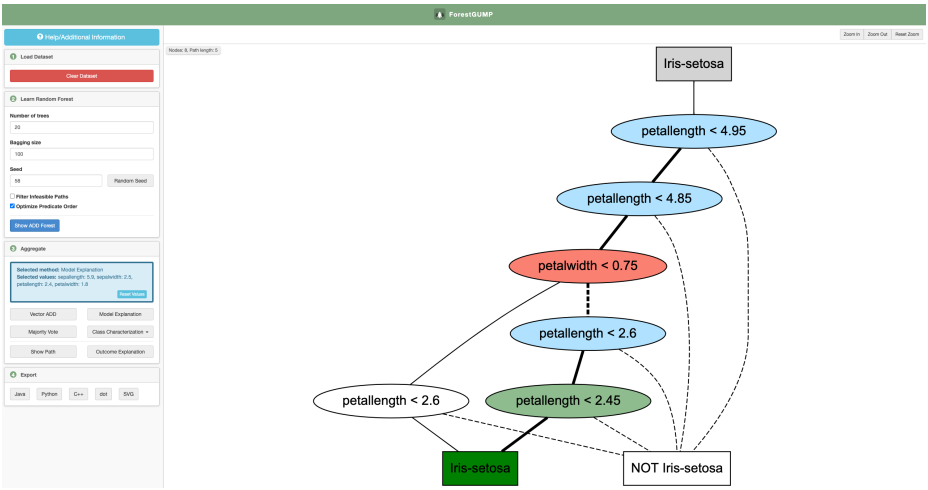

Fig. 14. The outcome explanation for the input $petallength = 2.4$, $petalwidth = 1.8$, $sepallength = 5.9$, $sepalwidth = 2.5$ (8 nodes, highlighted path of length 5) where the class characterization from Figure 11 is reordered.

explanation, *outcome explanation*, and *class characterization*. Key technology to achieve model explanation is algebraic aggregation, i.e. the transformation of a Random Forest into a semantically equivalent, concise white-box representation in terms of Algebraic Decision Diagrams. Class characterization is then achieved in terms of BDDs where the structure unnecessary to distinguish the considered class is collapsed. This abstraction is not only interesting in itself to better understand how easily the classes can be separated, but it also leads to highly optimized outcome explanations. Together with infeasible path elimination and the suppression of redundant predicates on a path, we observe reductions of outcome explanations by more than an order of magnitude. Forest GUMP allows even newcomers to easily experience these phenomena without much training.

Of course, these are first steps in a very ambitious new direction and it has to be seen how far the approach carries. Scalability will probably require decomposition methods, perhaps in a similar fashion as illustrated by the difference between model explanation and the considerably smaller class characterization. More work is needed also on techniques that aim at limiting the number of involved predicates.

Data Availability Statement: The artifact is available in the Zenodo repository [22].

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **27**(6), 509–516 (1978)
2. Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: Proceedings of 1993 Inter-

national Conference on Computer Aided Design (ICCAD). pp. 188–191 (1993). https://doi.org/10.1109/ICCAD.1993.580054
3. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (Oct 2001). https://doi.org/10.1023/A:1010933404324
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819
5. Chipman, H.A., George, E.I., McCulloch, R.E.: Making sense of a forest of trees (1999)
6. Deng, H.: Interpreting tree ensembles with intrees. Int. J. Data Sci. Anal. **7**(4), 277–287 (2019). https://doi.org/10.1007/s41060-018-0144-8
7. Domingos, P.M.: Knowledge discovery via multiple models. Intell. Data Anal. **2**(1-4), 187–202 (1998). https://doi.org/10.1016/S1088-467X(98)00023-7
8. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of eugenics **7**(2) (1936)
9. Gossen, F., Margaria, T., Murtovi, A., Naujokat, S., Steffen, B.: Dsls for decision services: A tutorial introduction to language-driven engineering. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Modeling - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11244, pp. 546–564. Springer (2018). https://doi.org/10.1007/978-3-030-03418-4_33
10. Gossen, F., Margaria, T., Steffen, B.: Towards explainability in machine learning: The formal methods way. IT Prof. **22**(4), 8–12 (2020). https://doi.org/10.1109/MITP.2020.3005640
11. Gossen, F., Margaria, T., Steffen, B.: Formal methods boost experimental performance for explainable AI. IT Prof. **23**(6), 8–12 (2021). https://doi.org/10.1109/MITP.2021.3123495, https://doi.org/10.1109/MITP.2021.3123495
12. Gossen, F., Murtovi, A., Linden, J., Steffen, B.: The java library for algebraic decision diagrams. https://add-lib.scce.info, accessed: 2022-01-13
13. Gossen, F., Steffen, B.: Large random forests: Optimisation for rapid evaluation. CoRR **abs/1912.10934** (2019), http://arxiv.org/abs/1912.10934
14. Gossen, F., Steffen, B.: Algebraic aggregation of random forests: towards explainability and rapid evaluation. International Journal on Software Tools for Technology Transfer (Sep 2021). https://doi.org/10.1007/s10009-021-00635-x
15. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. **51**(5), 93:1–93:42 (2019). https://doi.org/10.1145/3236009
16. Hara, S., Hayashi, K.: Making tree ensembles interpretable: A bayesian model selection approach. In: Storkey, A.J., Pérez-Cruz, F. (eds.) International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain. Proceedings of Machine Learning Research, vol. 84, pp. 77–85. PMLR (2018), http://proceedings.mlr.press/v84/hara18a.html
17. Ho, T.K.: Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition. vol. 1, pp. 278–282 vol.1 (1995). https://doi.org/10.1109/ICDAR.1995.598994
18. Hungar, H., Steffen, B., Margaria, T.: Methods for generating selection structures, for making selections according to selection structures and for creating selection descriptions. https://patents.justia.com/patent/9141708 (Sep 2015), USPTO Patent number: 9141708

19. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell System Technical Journal **38**(4), 985–999 (1959)
20. Lou, Y., Caruana, R., Gehrke, J.: Intelligible models for classification and regression. In: Yang, Q., Agarwal, D., Pei, J. (eds.) The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012. pp. 150–158. ACM (2012). https://doi.org/10.1145/2339530.2339556
21. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
22. Murtovi, A., Bainczyk, A., Steffen, B.: Forest gump: A tool for explanation (tacas 2022 artifact) (Nov 2021). https://doi.org/10.5281/zenodo.5733107
23. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
24. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. pp. 1135–1144. ACM (2016). https://doi.org/10.1145/2939672.2939778
25. Somenzi, F.: Cudd: Cu decision diagram package release 3.0 (2015)
26. Steffen, B., Gossen, F., Naujokat, S., Margaria, T.: Language-Driven Engineering: From General-Purpose to Purpose-Specific Languages, pp. 311–344. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_17
27. Van Assche, A., Blockeel, H.: Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In: Kok, J.N., Koronacki, J., Mantaras, R.L.d., Matwin, S., Mladenič, D., Skowron, A. (eds.) Machine Learning: ECML 2007. pp. 418–429. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
28. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edn. (2016)
29. Zhou, Y., Hooker, G.: Interpreting models via single tree approximation (2016)