Decision Snippet Features

Pascal Welke
University of Bonn
welke@cs.uni-bonn.de

Fouad Alkhoury
University of Bonn
Fraunhofer IAIS

Christian Bauckhage
University of Bonn
Fraunhofer IAIS

Stefan Wrobel
University of Bonn
Fraunhofer IAIS

***Abstract*—Decision trees excel at interpretability of their prediction results. To achieve required prediction accuracies, however, often large ensembles of decision trees – random forests – are considered, reducing interpretability due to large size. Additionally, their size slows down inference on modern hardware and restricts their applicability in low-memory embedded devices. We introduce *Decision Snippet Features*, which are obtained from small subtrees that appear frequently in trained random forests. We subsequently show that linear models on top of these features achieve comparable and sometimes even better predictive performance than the original random forest, while reducing the model size by up to two orders of magnitude.**

I. Introduction

Decision trees (DT) and random forests (RF) are among the most used machine learning models. They have many nice properties such as interpretability [14, 15] and rather small model size, at least compared to recent deep learning models. In particular, decision trees excel in interpretability, if they are not too large. This is due to the fact that the prediction for a given data point depends on a sequence of simple tests on data features. However, a single decision tree sometimes has too low predictive performance due to overfitting and pruning issues. Hence ensembles, i.e., random forests are very common in practice.

Random forests reduce the variance in predictions of a single decision tree, while typically only marginally increasing the bias [3]. However, the typically large size of those ensembles reduces the interpretability. Another drawback of the large size of random forest models is that inference on small embedded devices becomes more difficult. Here, energy consumption of the inference step, as well as tight constraints on instruction memory require small model sizes [5]. Furthermore, on modern general purpose processors, repeated highly unpredictable branching as is common in the inference step of random forest models is slow due to often failing branch prediction. As a result there has been a recently increased interest in implementing DT and RF models in dedicated hardware, such as field programmable gate arrays (FPGA) [5] or for fast hardware-specific implementations [6].

Hence, models of small size are beneficial for implementation, as well as interpretability, while ensemble methods are often required to achieve the predictive performance that is needed. In this work, we propose to learn from trained models, (i) to increase interpretability and gain more insights in the underlying learning problem and (ii) to subsequently reduce the size of the resulting models for implementation on small embedded devices.

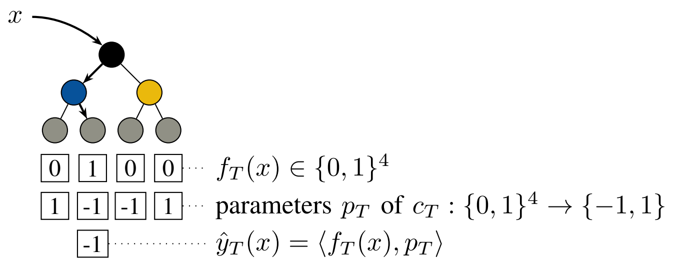
We propose to find rooted subtrees that appear frequently in a trained ensemble model. Due to the training process of random forests that use bootstrap aggregating (bagging), each tree in the forest model sees different data. Our working assumption is that certain combinations of splits that appear together in multiple trees are induced by the underlying learning problem. If this is true, such subtrees which we call *decision snippets* can be of independent interest for exploration of the problem domain. Furthermore, these decision snippets can be used in practice to build novel, smaller ensemble methods for the same learning problem.

To this end, we observe that a decision tree T is can be seen as learning a representation for the initial problem domain $\mathcal{X}$ that corresponds to the leaves of T [1]. Next follows a linear model (classification), piecewise linear model (regression), or nonlinear model (e.g., density estimation [8]) that maps leaves to the target domain. Thus, it is in principle possible to decouple the training process to find a suitable decision tree or random forest topology (i.e, a tree of split nodes and leafs) from the training process to find a suitable classification or regression model.

The view of decision tree or random forest learning as combination of representation and model learning allows to apply decision snippets as feature maps to obtain *decision snippet features* and to subsequently learn a suitable model on top of the novel feature representation. In this sense, each decision snippet corresponds to a multi-nominal feature in a novel feature space. In a second step, we then train linear classifiers in our novel feature space and show that they achieve similar predictive performance to the complete random forest model, while using only a small portion of the space or the original models. While in principle any model could be used on top of the decision snippet features, our goal of small and interpretable models can be well achieved with Naïve Bayes, linear Support Vector Machines, and Logistic Regression.

Our empirical evaluation shows that our decision snippet features allow similar or even better prediction accuracies on standard benchmark datasets while reducing the model size as well as inference time, compared to the original random forests that they were obtained from.

The remainder of this paper is organized as follows: In Section II, we introduce the necessary notions and notation. Section III motivates our usage of linear models on top of decision snippet features by showing that random forests, in fact are nothing else. Next, we introduce decision snippet



x
0 1 0 0 $f_T(x) \in \{0,1\}^4$
1 -1 -1 1 parameters p_T of $c_T : \{0,1\}^4 \to \{-1,1\}$
-1 $\hat{y}_T(x) = \langle f_T(x), p_T \rangle$

Fig. 1: A decision tree T with three split vertices (split conditions are depicted by colors) and four leaf vertices (depicted in gray). The inference step for T with majority classes $(1, -1, -1, 1)$ in its leaves for some input vector $x \in \mathbb{R}^d$ can be seen as a feature mapping step $f_T(x)$ followed by a linear function $y_T(f_T(x))$, resulting in the class prediction -1 in this example.

features with some technical detail in Section IV and show empirical results in Section V. Related work is discussed in Section VI before Section VII concludes.

II. Notions and Notation

We consider a learning problem with a hidden function $y : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{Y} \subseteq \mathbb{R}$ where we are given a set $X_{\text{train}} \subseteq \mathcal{X}$ and $y_{\text{train}} \in \mathcal{Y}^{X_{\text{train}}} : y_i = y(x_i)$ and our goal is to come up with a prediction function $\hat{y} : \mathcal{X} \to \mathbb{R}$ such that $\hat{y}(x)$ is close to $y(x)$ for all $x \in \mathcal{X}$. Many learning tasks can be modeled in this way, by defining a suitable feature representation for the objects of interest and by defining a sensible loss function to measure the closeness of $\hat{y}$ to y. A well known way to come up with a function $\hat{y}$ are decision trees and random forests, which we will now introduce.

A *rooted tree* is a tree $T = (V(T), E(T))$ in which one vertex $r(T) \in V(T)$ has been designated the *root*. We consider rooted trees as directed graphs where each edge is directed away from the root vertex and denote an edge from v to w as (v, w). Hence each vertex $v \in V(T)$ has a unique depth $d(v)$ which is its distance from the root $r(T)$. A *binary rooted tree* is a rooted tree such that every vertex has either zero or two outgoing edges. The set of leaves of T is the set of all vertices in T with no outgoing edges. A *labeled rooted tree* over some set Σ, called label set, is a rooted tree T together with a label function $l_T : V(T) \cup E(T) \to \Sigma$. A labeled rooted tree T_1 is *subgraph isomorphic* to a labeled rooted tree T_2 if there exists an injective mapping $\varphi : V(T_1) \to V(T_2)$, called *subgraph isomorphism* such that $l_{T_1}(v) = l_{T_2}(\varphi(v))$ and $l_{T_1}((v, w)) = l_{T_2}((\varphi(v), \phi(w))$ for all $v, w \in V(T_1)$ and $d(\varphi(r(T_1))) \leq d(\varphi(v))$ for all $v \in V(T_1)$. That is, if we draw rooted trees with root on top, the image of the root of T_1 in T_2 will be on top of the image of T_2. We will omit the subscripts from the notation if clear from the context.

With these definitions, we consider a random forest model F for a numerical domain set $\mathcal{X} \subseteq \mathbb{R}^d$ and a response variable set $\mathcal{Y}$ to be a multi set of rooted labeled binary trees. Here, each label on a vertex corresponds to a split condition $x_i \leq v$ for some $i \leq d$ and $v \in \mathbb{R}$. Each label on an edge corresponds to either "the split condition evaluates to *true*" or "the split condition evaluates to *false*". Note that we can model categorical domain sets as subsets of $\mathbb{R}^d$, as well by using one-hot encodings. In this paper, we will always assume that we are given a decision tree or random forest model F that was learned on some training data set $X_{\text{train}} \subset \mathbb{R}^d$ and will denote the corresponding prediction function by $\hat{y}_F$.

III. Random Forests are Linear Models

Recently there has been a tremendous amount of research in the area of representation learning. Most of the work employs (deep) neural networks to simultaneously learn "useful" representations (i.e., features) and a corresponding prediction function $\hat{f}$ for a given learning problem [1, 11]. In this sense, the hidden neurons of a neural network can be interpreted as features (or sets of features, if tensors are propagated through the neural network) and the output neurons are interpreted as the target features. The training procedures (typically some form of gradient descent) simultaneously builds features (hidden neurons) and target values (output neurons) that solve the task at hand. A typical problem of these approaches, however, is that the topology of the neural network has to be predefined. Decision trees and Random Forests, on the other hand, allow to build a topology for a given learning problem and, as we will discuss now, are indeed also learning a feature representation and corresponding prediction function.

For ease of exposition, let us consider the case of binary classification, where the response variable takes values in $\mathcal{Y} = \{-1, 1\}$. Suppose we have learned a decision tree T on some training dataset $X_{\text{train}} \subseteq \mathbb{R}^d$. Classification is then typically done using the majority class or the relative frequency of the positive class of the training examples that were mapped to a leaf l in the training process. That is, conceptually, the inference step first maps an example x to a leaf of T and then returns the value associated with that leaf. Hence we can write the classification function $\hat{y}_T : \mathbb{R}^d \to \{-1, 1\}$ that corresponds to T as

$$\hat{y}_T = c_T \circ f_T$$

with

- $f_T : \mathbb{R}^d \to \text{leaves}(T)$ which maps an example $x \in \mathbb{R}^d$ to the leaf $l \in \text{leaves}(T)$ that the inference step of the decision tree classifier computes.
- $c_T : \text{leaves}(T) \to \{-1, 1\}$ maps a leaf l to the majority class of training examples assigned to l.

See Fig. 1 for an example of a decision tree with three internal split vertices and four leaf vertices that predicts $\hat{y}_T(x) = -1$.

Now, y_T is in fact a linear function in the following sense: We can represent $\text{leaves}(T)$ as a one-hot encoded feature space, i.e., $f_T : \mathbb{R}^d \to \{0, 1\}^{\text{leaves}(T)}$ with $f_T(x) = e_{l(x)}$ where e_l is the unit vector that is equal to one at index l and equal to zero otherwise and $l(x)$ is the leaf of T which x is assigned to. That is, each $x \in \mathbb{R}^d$ is mapped to a unit vector in $\mathbb{R}^{\text{leaves}(T)}$. Now we can multiply this unit vector $e_{l(x)}$ with a weight vector $p_T \in \mathbb{R}^{\text{leaves}(T)}$ that contains, for every leaf, its majority class. Hence the class of an input example $x \in \mathbb{R}^d$ can be obtained by computing the scalar product $\langle f_T(x), p_T \rangle$, which is a linear function for fixed p_T.[1]

If instead of a single decision tree we consider an ensemble of decision trees $F = \{T_1, \dots, T_k\}$, we can concatenate the feature maps f_{T_i} to a feature map

$$f_F : \mathbb{R}^d \to \{0,1\}^{\text{leaves}(T_1) \times \ldots \times \text{leaves}(T_k)} \quad (1)$$
$$x \mapsto f_{T_1}(x) \ldots f_{T_k}(x) \quad (2)$$

for the ensemble F.

Two approaches for inference on a random forest are most common: (1) Averaging and (2) majority voting. Both variants can directly be achieved by a linear classifier on the feature values f_F. For averaging, we set $p_F \in \mathbb{R}^{\text{leaves}(T_1) \times \ldots \times \text{leaves}(T_k)}$ to store the probabilities of the *positive* class for each leaf, and using a threshold value of 0.5 for the linear classifier. Majority voting can be achieved by setting p_F to store the *majority* class for each leaf and choosing 0 as threshold value for the linear classifier.

Furthermore, a decision tree T that stores the positive class probability in p_T solves the same task that a Naïve Bayes classifier solves. That is, $\langle p_T, f_T(x) \rangle = \mathbb{P}(\hat{y}(x) = 1 | x) = p_{l(x)}$ and the linear classifier mentioned above returns $\arg\max_{c \in \{-1,1\}} \mathbb{P}(c | f_T(x))$. The classification step of a random forest, however, is different to Naïve Bayes.

These connections motivate us to apply linear classifiers on top of our decision snippet features. In particular, we will investigate linear support vector machines, logistic regression, and Naïve Bayes models on top of the features f_F. While the training process obviously differs from that of a random forest (which learns the weights p_F together with the topology), the resulting models are similarly easy to interpret, as well as implement and require the same amount of parameters.

IV. Decision Snippet Features

Recall that we are interested in small models due to their inherent interpretability, fast inference, and applicability on small embedded devices. If random forest inference is a linear model on a non-linear feature space then we can in principle separate the feature selection step from the classifier training process. With this newly gained freedom we shall investigate how to construct a novel feature mapping which we call *decision snippet features* from a trained random forest. To this end, we start with a look at the split conditions.

A split condition that appears frequently in a trained random forest has been selected by the training process of a decision tree algorithm on multiple subsets of the data because it has maximized some relevant gain function (or minimized some impurity measure). Note that this is true not only because each decision tree in a random forest is trained on a different subset of the original training data. It is also due to the fact that a split vertex that is not the root of its decision tree only sees a

[1]The same method works for class probabilities stored in the leaves. It is also easy to see how to generalize this to non-binary classification or simple variants of regression trees.

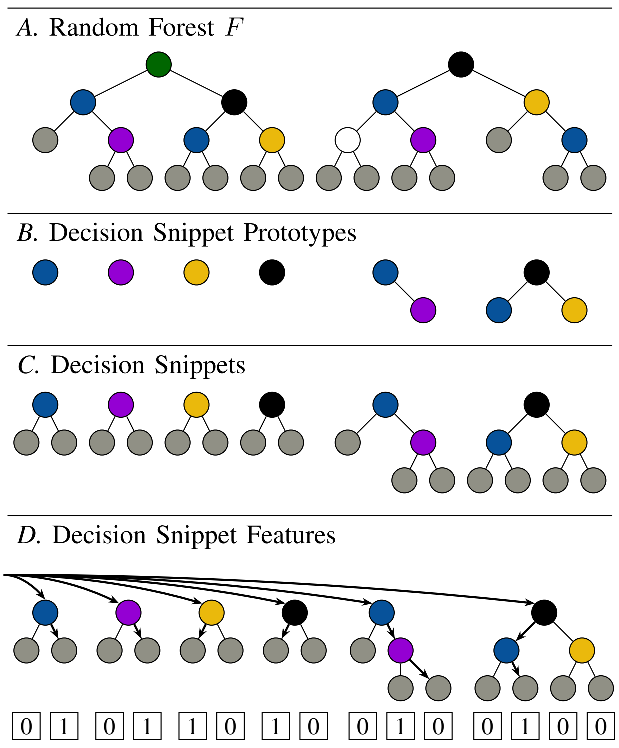


Fig. 2: *A.* A random forest F consisting of two decision trees (split conditions are depicted by colors), *B.* all 2-frequent subtrees of the random forest (without leaves) selected as decision snippet prototypes, *C.* the resulting decision snippets, and *D.* the resulting decision snippet features. Gray vertices are leaves, all other colors represent unique branching conditions, such as $f_2 \leq 42$. The decision snippet features which result from the decision snippets map an input example x to one leaf in each decision snippet, resulting in a 15-dimensional binary decision snippet feature vector. This vector always has exactly as many ones as there are decision snippets.

subset of this random subset of the original training data. This extends beyond single split conditions: A rooted labeled tree that appears more than once in a random forest tells us that a sequence of tests maximally increases the gain function on multiple subsets of the training data. We will ignore the leaves of the decision tree for now, and will motivate this soon.

We call such subtrees of a random forest *decision snippet prototypes* and are interested in finding them. Before we go into the technical details of this, however, we will explain how to use decision snippets to achieve our goal of small, fast, interpretable models. A decision snippet prototype S is almost a decision tree. However, S might contain split vertices as leaves. Consider, for example, the yellow vertex in the random forest on top of Fig. 2. This vertex appears once directly above two leaf vertices and once in a higher level of the decision tree on the top right. So we should select the yellow split condition as a decision snippet prototype. To obtain a proper decision tree from it, however, we need to add two leaves and connect them via edges that correspond to the two possible outcomes of the yellow split condition test. The resulting decision tree S' can now be used to compute a leaf (of S') for any data point $x \in \mathbb{R}^d$. Hence we can apply this methodology for any set $P = \{S_1, \ldots, S_l\}$ of decision snippet prototypes to obtain a set $P' = \{S'_1, \ldots, S'_l\}$ of *decision snippets* and define the feature map $f_{P'}$ as in Eqn. 1. We call the resulting features *decision snippet features*. Figure 2 shows the whole process for a random forest F consisting of two decision trees.

As a result, we have now effectively constructed a new random forest that can be used for mapping an input vector $x \in \mathbb{R}^d$ to a one-hot encoded feature vector, the decision snippet features. What now remains is to learn the function $y_{P'}$ to map from the decision snippet feature space to the target domain: We cannot just take the p_F values from the original random forest. First, we might have constructed new leaf nodes for some decision snippet prototypes. Second, even if we had selected a leaf node as part of a decision snippet prototype, we could not use its corresponding majority class (or its probability). The reason is that the decision snippet will see all input data during inference, while the corresponding subtrees in the random forest F would only see a subset of the data, depending on the split conditions on the parent vertices. Hence we have to learn a new function.

In principle, any learning method could be used. However, we will stick to simple linear models, as these can be implemented on restricted hardware and maintain the interpretability and small size of the resulting models.

A. Technical Details and Computational Complexity of Mining Decision Snippet Features

To obtain decision snippet features, we propose to select a set P of (small) rooted subtrees that appear frequently in a trained random forest F. That is, given F and a frequency threshold $t \in \mathbb{N}$ we enumerate all rooted labeled trees that are subgraph isomorphic to at least t (decision) trees in F. t is a parameter that can be chosen by the user or found by some parameter optimization method. See Fig. 2 for an example random forest F and the 2-frequent subtrees of F (ignoring the leaves; more on this later).

All frequent subtrees in the above sense can be found in polynomial time in the size of the random forest F *and the number of such frequent subtrees* using some off-the-shelf frequent subtree mining algorithms [7]. These algorithms typically proceed by finding all frequent labeled vertices first (i.e., the split conditions that appear frequently) and then extending them one by one with an additional (frequent) split condition that is connected either by a "true" edge, or a "false" edge. Subsequently, the support of such a candidate pattern is evaluated on the random forest. For a recent overview on such algorithms see Welke [16]. We note that the special case of binary random forests allows for faster subtree isomorphism checks, as they are in fact ordered rooted trees with bounded out-degree. But we omit the discussion of these details as the typical size of the random forests does not require a very careful algorithmic approach: Even very large random forests are small when considered as frequent subtree mining instances.

Dataset	# Instances	# Attributes	# Classes
adult	48842	14	2
bank	45211	17	2
magic	19020	11	2
spambase	4601	57	2
statlog	6435	36	7
mnist	60000	64	10
sensorless	58509	49	11
letter	20000	16	26

TABLE I: Datasets. We report number of instances, number of attributes, and number of classes.

As we have mentioned above, after obtaining a set of frequent subtrees P of a random forest F as decision snippet prototypes, we next need to add missing leaves. This step, however, might result in duplicates in the resulting set of decision snippets: Consider a decision snippet prototype S_i that contains (by chance) some leaf vertex v. As P is the set of frequent subtrees, it follows that P also contains the tree S_j that is obtained from S_i by removing the leaf v. This is due to the downward closure property of P: If some tree S is frequent in F, all of its subtrees must be frequent in F, as well. But adding all missing leaves to S_i and to S_j would result in two isomorphic trees, resulting in redundant decision snippet features and a larger model size. To avoid this issue, we remove the leaves from the random forest F before mining the frequent subtrees. It can be shown that this exactly removes all duplicate decision snippets, without losing any non-duplicate.

V. Experiments

We show experiments on eight benchmark classification datasets from the UCI repository [9] (adult, spambase, magic, satlog, mnist, letter, bank, and sensorless) containing both categorical and numerical features. We train four random forest classifiers on each dataset for a maximum depth $d \in \{5, 10, 15, 20\}$, consisting of 25 trees each. We use the standard train/test split of the datasets and use a standard implementation of random forests [13] with default parameters. See Table I for information on the datasets.

To obtain frequent subtrees in these random forests, we use a rooted version of the frequent subtree mining software described in Welke [16]. To this end, we convert each random forest into a rooted labeled tree database where edges are either labeled "true" or "false" and vertices have a label of the form "$f_i \leq v$" for some string representation of feature f_i and fixed precision representation of v. As discussed in Section IV-A, we remove all leaf vertices from the random forests to speed up the mining and avoid duplicate decision snippets. We compute the set of frequent subtrees in these datasets for each frequency threshold $t \in [2, 25]$, which make up the decision snippet candidates. To restrict the runtime and space requirements of the resulting decision snippet features, we restrict the mining step to patterns of up to at most six vertices. To allow subsequent experimentation with scikit-learn tools, we provide a compatible class in Python that takes the decision snippet candidates, adds leaves where they are missing and can compute the decision snippet features for some given set of examples from the input domain $\mathcal{X} \subseteq \mathbb{R}^d$. Our code and experiments are available online.[2]

We train Naïve Bayes (NB), logistic regression (LR) and linear support vector machine (SVM) models on the decision snippet features and compare their performance to the perfomance of the random forest on the original feature representation. For training the models on the decision snippet features, we reuse the training split of the dataset. Among the $4 \cdot 24$ sets of decision snippet feature candidates for each dataset, we select te best candidate as decision snippet features. Selection of the best decision snippet features is based on the accuracy on the training split. We report the accuracy on the test set on all subsequent tables and figures.

Dataset	Random Forest			Decision Snippet Features			
	Acc.	Size	Avg.I	θ	Acc.	Size	Avg.I
adult	0.863	99263	406.7	2	0.843	176	153.8
spambase	0.93	10183	249.5	3	0.922	582	199
magic	0.856	69961	409.9	2	0.845	1192	266.6
satlog	0.861	17473	298.3	2	0.871	141	138.4
mnist	0.966	237187	398.1	4	0.872	305	199
letter	0.961	92323	340.2	2	0.932	23220	380.9
bank	0.501	1479	125	2	0.905	144	134
sensorless	0.11	8889	250	2	0.102	60	59

TABLE II: Summary of Table IV. Best (according to test accuracy) Random Forest of any maximum depth $d \in \{5, 10, 15, 20\}$ and best Decision Snippet Features with their sizes and average inference complexities (Avg.I) on test set. θ reports the frequency threshold used for the best decision snippets.

A. *Size, Inference Complexity, and Predictive Performance*

To investigate whether decision snippet features are indeed suitable for reducing the model size (e.g., for inference on embedded devices) we compute the size of the original random forests and the decision snippet features. We report the number of nodes (i.e., split conditions and leaves) of the decision snippets and random forests. To complement these numbers, we also compute the inference complexity of both models, that is, the average number of comparisons (if conditions) required for a test example to pass from the root of the tree until reaching the prediction in one of the leaf nodes.

Table IV shows a comparison between the complete Random Forest (consisting of 25 trees each) and the decision snippet features according to accuracy, size and inference complexity for each depth $d \in \{5, 10, 15, 20\}$. Table II) summarizes Table IV, showing only the best random forest and the best decision snippet features among all depths $d \in \{5, 10, 15, 20\}$. Starting with Tab. II, it can be seen that the best performing learners on decision snippet features are typically within 0.02 of the accuracy of the best random forest. Exceptions are only mnist, where decision snippets don't seem to work well, and bank, where the random forests do not perform better than random guessing, but nonetheless the best decision snippet based classifier achieves an accuracy of 0.905. On the other hand, when looking at the size of the models, we see that the decision snippet features are between two and three orders of magnitude smaller (in number of nodes) than the best performing random forests. The average inference time of decision snippet features, (as measured by the average number of if-conditions to be evaluated for inference) is on average 0.65 that of the best performing random forest. We must acknowledge, however, that there are cases, where the average inference time increases a little, due to a large number of small decision snippets.

Looking at the detailed results in Table IV, we see that on the decision snippet features, Naïve Bayes is typically outperformed by either logistic regression or linear SVM. Furthermore, it can be seen that for all depths, the decision snippets reduce the model size, with respect to the random forest from which they were mined. This is not necessarily the case, as there might be an exponential number of frequent patterns for unconstrained tree databases. However, the frequency thresholds for which the (small) best performing models are found tends to be rather low (between two and five) with a few exceptions.

[2]https://github.com/pwelke/DecisionSnippetFeatures

B. *Is Frequency a Good Selector of Decision Snippet Features?*

We now empirically validate our assumption that frequent subtrees of trained random forests are indeed good predictors. To this end, we compare the performance of (frequency based) decision snippet features to the performance of subtrees sampled randomly from the random forest. For a given dataset D and random forest depth $d \in 5, 10, 15, 20$ we compute the set of all 1-frequent subtrees (i.e. all trees that occur anywhere in the database regardless of the their frequency) up to six vertices. Then we sample the same number of patterns from the resulting set of subtrees as in the best performing frequency based decision snippet candidate of the same maximum depth d. We build the decision snippet features from this new set of trees and proceed with the training as for the frequency based decision snippet features. We repeat the sampling procedure ten times and report average and standard deviation over the random samples.

Table III compares results of this experiment. We first note that the above sampling process draws uniformly from the set of all subtrees of size up to six (excluding leaves), while the frequency based selection chooses only frequent subtrees. Due to the downward closure of the pattern set the latter favors smaller patterns. Hence we notice that the sampling based decision snippet features have on average 4.5 times more vertices than the frequency based decision snippet features. Furthermore, the size of the sampling based decision snippet features does not deviate much among the different samples.

When comparing the prediction accuracies of frequency based and sampling based decision snippet features, we see that random sampling tends to perform better on magic, satlog, mnist, and letter, while the opposite is true on the remaining

Dataset	Frequency Based			Sampling Based	
	θ	Acc.	Size	Acc.	Size
		Logistic Regression			
adult	2	0.83	176	0.825 ± 0.004	644 ± 14.3
spambase	2	0.794	81	0.624 ± 0.071	355.5 ± 12.3
magic	2	0.825	172	0.843 ± 0.002	750.9 ± 27.1
satlog	2	0.868	141	0.876 ± 0.003	646.8 ± 19.7
mnist	2	0.712	86	0.917 ± 0.003	408.7 ± 13.6
letter	3	0.85	95	0.866 ± 0.009	416.5 ± 12.2
bank	2	0.905	144	0.908 ± 0.001	611.4 ± 21.7
sensorless	2	0.091	11	0.093 ± 0.001	51.1 ± 5.5
		Naïve Bayes			
adult	2	0.843	176	0.728 ± 0.019	644 ± 14.3
spambase	2	0.904	81	0.899 ± 0.024	355.5 ± 12.3
magic	5	0.778	24	0.789 ± 0.009	116.2 ± 5.1
satlog	2	0.764	141	0.759 ± 0.056	646.8 ± 19.7
mnist	2	0.712	86	0.786 ± 0.009	408.7 ± 13.6
letter	5	0.5	54	0.529 ± 0.015	265 ± 7.5
bank	14	0.896	4	0.745 ± 0.1	19.9 ± 2.7
sensorless	2	0.091	11	0.093 ± 0.002	51.1 ± 5.5
		SVM			
adult	2	0.824	176	0.826 ± 0.004	644 ± 14.3
spambase	4	0.791	8	0.692 ± 0.053	37.4 ± 4.9
magic	2	0.824	172	0.842 ± 0.002	750.9 ± 27.1
satlog	2	0.871	141	0.87 ± 0.004	646.8 ± 19.7
mnist	2	0.855	86	0.916 ± 0.003	408.7 ± 13.6
letter	3	0.847	95	0.873 ± 0.01	416.5 ± 12.2
bank	3	0.904	75	0.907 ± 0.001	356 ± 10.2
sensorless	8	0.098	1	0.102 ± 0.002	4.6 ± 1.4

TABLE III: Comparison of frequency based decision snippet features (left) and randomly sampled decision snippet features (right). We report mean and std. dev. of ten random samples.

four datasets. Reconsidering the size of the models, however, we see that (except for mnist) frequency based decision snippet features perform at most slightly worse than randomly sampled decision snippet features, using only 23% of the required space.

C. *Using Decision Snippet Features as Random Forests*

In the previous steps, we have applied linear models on top of decision snippet features. Decision snippets, however, can also be used as random forests. To this end, recall from Section III that the random forest classifier computes the majority class of the training labels for each leaf vertex and then (in its basic versions) either does a majority voting or an averaging over the decision trees in the random forest. Hence, we compute training set class probabilities for each leaf of any decision snippet and then set the leaf label to the majority class (majority voting) or the class with the higher sum of probabilities in the respective leaf vertices (averaging) at inference time.

Figure 3 shows the results of this exeriment. We deliberately compare the results of the Naïve Bayes classifier on the decision snippet features, as it performed the worst among the three classifiers considered in Sect. V-A. However, the Naïve Bayes classifier outperforms the majority voting and averaging classifiers on six out of the eight datasets under consideration and achives similar results on the remaining two datasets. Hence, we conclude that decision snippets are not well suited to directly work as random forests. This is not

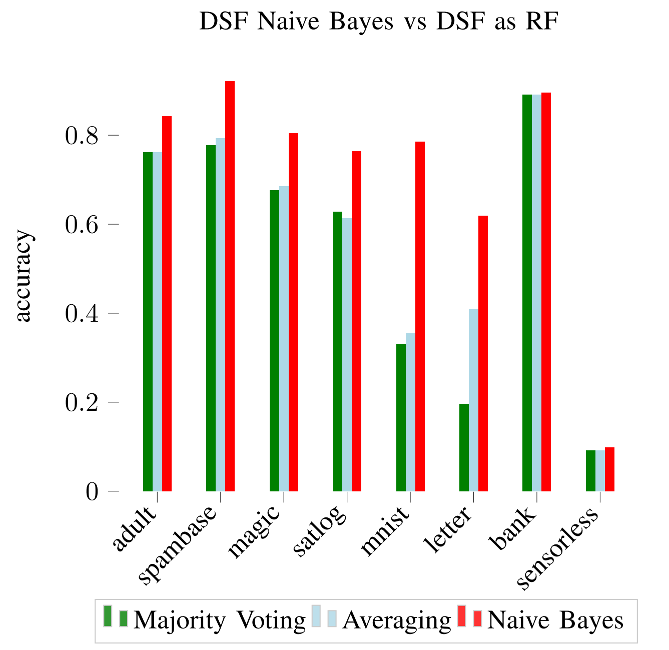


Fig. 3: The plot shows the performance comparison between DSF Naive Bayes and the DSF as Random Forest (using two variants: majority voting and average probability)

surprising as the bagging in combination with pruning might have otherwise resulted in a random forest of similar size and topology.

VI. Related Work

A. *Model Trees*

Model trees [10] use linear regression functions associated with each leaf of a decision tree to learn a piecewise linear function. They can be used for classification, as well. Note, however, that our approach is conceptually simpler, as our approach builds a single linear function from the leaf space to the target domain.

B. *Representation Learning*

The fact that decision trees are implicitly learning a one-hot encoded representation of contiguous regions in input space has been observed before [1]. In this context, Bengio and Delalleau [2] observed that the expressive power of ensemble models, that is random forests, is much higher than that of a single decision tree. This is due to a possibly exponential blow-up of the intersections of regions of different decision trees in the random forest model. While the thought of mining relevant substructures in trained random forests is (up to our knowledge) novel, this motivates our search for reasonable representations in the trained random forest and the subsequent application of a suitable model.

C. *Pruning*

There has been a tremendous amount of work dedicated to the pruning of decision trees. Even the initial works of Breiman et al. [4] and Quinlan [14] mention the necessity

Dataset	Random Forest			DSF Naive Bayes				DSF Logistic Regression				DSF SVM			
	Acc.	Size	Avg.I	θ	Acc.	Size	Avg.I	θ	Acc.	Size	Avg.I	θ	Acc.	Size	Avg.I
Max depth = 5															
adult	0.844	1431	124.1	**2**	**0.843**	**176**	**153.8**	**2**	**0.830**	**176**	**153.8**	2	0.824	176	153.8
spambase	0.915	1205	124.2	2	0.904	81	78.3	2	0.794	81	78.3	4	0.791	8	7
magic	0.820	1471	125	5	0.778	24	23	2	0.825	172	163.3	2	0.824	172	163.3
satlog	0.841	1469	122.7	**2**	**0.764**	**141**	**138.4**	**2**	**0.868**	**141**	**138.4**	**2**	**0.871**	**141**	**138.4**
mnist	0.856	1575	125	2	0.712	86	85	2	0.855	86	85	2	0.854	86	85
letter	0.659	1439	124.7	5	0.497	54	53	3	0.850	95	87.7	3	0.847	95	87.7
bank	**0.501**	**1479**	**125**	**14**	**0.896**	**4**	**3**	**2**	**0.905**	**144**	**134**	**3**	**0.904**	**75**	**74**
sensorless	0.109	1173	125	2	0.091	11	10	2	0.097	11	10	8	0.098	1	0
Max depth = 10															
adult	0.859	14705	244	7	0.834	145	143.3	8	0.826	130	129	9	0.825	125	124
spambase	0.926	5929	187.8	4	0.916	83	82	9	0.777	3	2	9	0.777	3	2
magic	0.845	13573	200	**2**	**0.804**	**1192**	**266.6**	**2**	**0.845**	**1192**	**266.6**	**2**	**0.844**	**1192**	**266.6**
satlog	0.857	10345	224.2	4	0.753	501	199	2	0.860	1223	204.7	2	0.858	1223	204.7
mnist	0.944	41993	249.2	**3**	**0.785**	**841**	**199**	**4**	**0.872**	**305**	**199**	**4**	**0.869**	**305**	**199**
letter	0.865	17923	234.5	4	0.580	345	243.4	4	0.875	345	243.4	4	0.875	345	243.4
bank	0.458	18105	248.1	25	0.865	30	29	5	0.904	303	225.6	5	0.904	303	225.6
sensorless	**0.110**	**8889**	**250**	2	0.097	60	59	**2**	**0.102**	**60**	**59**	**2**	**0.101**	**60**	**59**
Max depth = 15															
adult	0.862	50719	337.7	3	0.803	3433	327.9	19	0.824	128	127	19	0.825	128	127
spambase	**0.93**	**10183**	**249.5**	6	0.919	60	59	13	0.777	3	2	13	0.777	3	2
magic	0.852	41633	344.5	2	0.793	2519	305.2	3	0.843	1150	232.9	3	0.843	1150	232.9
satlog	0.858	16371	257.8	9	0.721	129	128	2	0.860	1545	217	2	0.857	1545	217
mnist	0.962	186807	346.7	3	0.766	8388	200	2	0.868	20554	200	2	0.865	20554	200
letter	0.948	56379	309.1	7	0.581	335	240.5	2	0.913	13773	368.9	2	0.919	13773	368.9
bank	0.424	73195	348.3	4	0.769	4212	322.6	4	0.904	4212	322.6	4	0.903	4212	322.6
sensorless	0.109	27369	348.7	**2**	**0.098**	**283**	**199**	2	0.101	283	199	2	0.101	283	199
Max depth = 20															
adult	**0.863**	**99263**	**406.7**	13	0.774	346	248	22	0.826	151	149.9	**17**	**0.827**	**210**	**203.2**
spambase	0.93	12589	285.3	**3**	**0.922**	**582**	**199**	**2**	**0.821**	**1351**	**202.1**	**2**	**0.817**	**1351**	**202.1**
magic	**0.856**	**69961**	**409.9**	2	0.785	4987	307.5	2	0.836	4987	307.5	2	0.835	4987	307.5
satlog	**0.861**	**17473**	**298.3**	10	0.716	110	109	10	0.860	110	109	9	0.857	169	168
mnist	**0.966**	**237187**	**398.1**	3	0.721	11864	200	5	0.854	3049	200	5	0.853	3049	200
letter	**0.961**	**92323**	**340.2**	**2**	**0.619**	**23220**	**380.9**	**2**	**0.92**	**23220**	**380.9**	**2**	**0.932**	**23220**	**380.9**
bank	0.414	151685	435.2	4	0.816	8437	339.9	21	0.905	309	251.1	21	0.903	309	251.1
sensorless	0.106	34011	301.3	3	0.097	50	49	2	0.099	419	199	2	0.010	419	199

TABLE IV: Comparison of Random Forest and Decision Snippet Features. Shown are accuracy on test set (Acc.), node count (size) and inference complexity (Avg.I) for random forests of maximum depth $d \in \{5, 10, 15, 20\}$ and best decision snippet features mined from that depth. θ reports the frequency threshold used for the best decision snippets. Numbers in bold show the highest test set accuracy per dataset among random forests, and decision snippet features.

of pruning decision trees. See, e.g.,] for surveys on pre and postpruning. While decision snippets can be seen as an (extreme) form of pruning due to the size reduction, we note that typical pruning approaches map a single tree to a single tree (with fewer vertices) and maintain the inference scheme of the original decision tree, while our decision snippet features allow to extract multiple decision snippets per decision tree in a random forest.

VII. Concluding Remarks

We have proposed decision snippet features which are defined based on frequent subtrees in trained random forests. The empirical results on standard benchmark datasets indicate that they typically achieve predictive performances comparable to the original random forests, at a fraction of the size. This increases the interpretability of the resulting model and allows application in embedded devices with strict memory and power constraints.

We expect that our work can benefit from post-processing of random forests to increase their regularity. Recently, Nakamura and Sakurada [12] have given an algorithm that reduces the number of distinct branching conditions in random forests on real-valued domains. Such an approach would likely increase the probability of large frequent subtrees. Whether the combination of the two approaches results in higher accuracy decision snippet features is left for future work.

Acknowledgments

This research was partially funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01—S18038CB) and was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2070 – 390732324.

References

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 35, no. 08, pp. 1798–1828, aug 2013.

[2] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *International Conference*

on Algorithmic Learning Theory ALT, Proceedings, ser. Lecture Notes in Computer Science, J. Kivinen, C. Szepesvári, E. Ukkonen, and T. Zeugmann, Eds., vol. 6925. Springer, 2011, pp. 18–36. [Online]. Available: https://doi.org/10.1007/978-3-642-24412-4_3

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[5] S. Buschjäger and K. Morik, "Decision tree and random forest implementations for fast filtering of sensor data," *IEEE Transactions on Circuits and Systems*, vol. 65-I, no. 1, pp. 209–222, 2018.

[6] S. Buschjäger, K. Chen, J. Chen, and K. Morik, "Realization of random forest for real-time evaluation through tree framing," in *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*. IEEE Computer Society, 2018, pp. 19–28. [Online]. Available: https://doi.org/10.1109/ICDM.2018.00017

[7] Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok, "Frequent subtree mining - an overview," *Fundamenta Informaticae*, vol. 66, no. 1–2, pp. 161–198, 2005.

[8] C. Cousins and M. Riondato, "Cadet: interpretable parametric conditional density estimation with decision trees and forests," *Machine Learning*, vol. 108, no. 8-9, pp. 1613–1634, 2019. [Online]. Available: https://doi.org/10.1007/s10994-019-05820-3

[9] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[10] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten, "Using model trees for classification," *Machine Learning*, vol. 32, no. 1, pp. 63–76, 1998. [Online]. Available: https://doi.org/10.1023/A:1007421302149

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] A. Nakamura and K. Sakurada, "An algorithm for reducing the number of distinct branching conditions in a decision forest," in *European Conference on Machine Learning and Principles of Knowledge Discovery from Databases (ECMLPKDD) Proceedings*, 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: https://doi.org/10.1023/A:1022643204877

[15] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[16] P. Welke, "Efficient frequent subtree mining beyond forests," Ph.D. dissertation, University of Bonn, 2018.