Improved Feature Importance Computation for Tree Models Based on the Banzhaf Value

Abstract

The Shapley value – a fundamental game-theoretic solution concept – has recently become one of the main tools used to explain predictions of tree ensemble models. Another well-known game-theoretic solution concept is the Banzhaf value. Although the Banzhaf value is closely related to the Shapley value, its properties w.r.t. explaining predictions have not been understood equally well. This paper shows that, for tree ensemble models, the Banzhaf value offers some crucial advantages over the Shapley value while providing essentially the same explanations.

1 INTRODUCTION

Tree ensembles are one of the most commonly used models for solving practical problems [Friedman, 2001, Kaggle, 2017]. Tree ensembles are robust, easy to tune, and fast to train. They need small computational resources and support different types of data and missing values. Given this, tree ensembles are often the first choice model for tabular data.

One of the key research challenges regarding tree ensemble models and other machine learning techniques, in general, is explainability. When high-value decisions are taken, e.g., in medical diagnostic, understanding why a model made a specific prediction is often more important than the prediction's accuracy. Thus we need to develop methods to interpret the model's results in a transparent way so that humans are willing to follow model recommendations. As a result, recently, there has been a growing interest in the feature attribution problem, where, given some specific input x of features, one would like to attribute the model's prediction $f(x)$ to the individual features.

One of the most popular approaches to interpreting model predictions uses methods originating from cooperative game theory that are called solution concepts or *values*. They measure the importance of each player in, or contribution to, a coalitional game. While there exist many ways in which the importance of each player can be evaluated, some solution concepts are considered more fundamental than others due to underlying axiom systems that uniquely determine them. One important game-theoretic solution concept that attracted a lot of attention in the context of explainability is *the Shapley value* (e.g., [Lundberg et al., 2020, Lundberg and Lee, 2017, Štrumbelj and Kononenko, 2014, Sundararajan et al., 2017]). To formally introduce this concept, let us denote by $\langle g, U\rangle$, a coalition game where $g : 2^U \to \mathbb{R}$, $g(\emptyset) = 0$, is the *set function* that assigns utility to each coalition, and $U = \{1, \ldots, n\}$ is the set of players (or – in our context – features). Then, the Shapley value of the feature $i \in U$ is defined as follows:

$$\phi_i = \frac{1}{n} \sum_{S \subseteq U \setminus \{i\}} \binom{n-1}{|S|}^{-1} \left(g(S \cup \{i\}) - g(S)\right). \quad (1)$$

To operationalize this formula in our context, we further need to define function g that extends model f to all subsets of features $S \subseteq U$, i.e., g allows us to drop features $U \setminus S$ of both the input x and the model f. There are multiple alternatives of how this can be done proposed in the literature [Sundararajan and Najmi, 2020, Janzing et al., 2020]. In this paper, we focus on a popular approach by Lundberg and Lee [2017] (see Section 2 for more details). Furthermore, it should be noted that, while the Shapley value has certain attractive properties, it is evident from the above formula that, in the general case, it requires the input of exponential size (i.e., function g). However, in certain structured environments, when g is of a convenient form or is limited in size, the Shapley value can be computed in time polynomial in the number of players (features) [Deng and Papadimitriou, 1994, Michalak et al., 2013, Maafa et al., 2018] – e.g., for tree ensemble models [Lundberg et al., 2020].

However, the Shapley value is not the only solution concept that has been advocated for interpreting model predictions. Some papers suggest using the *Banzhaf value* for this pur-

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

pose [Datta et al., 2015, Patel et al., 2020, Sliwinski et al., 2019]. This value, also well-known and axiomatized, aggregates contributions of individual features differently:

$$\beta_i = \frac{1}{2^{n-1}} \sum_{S \subseteq U \setminus \{i\}} \left(g(S \cup \{i\}) - g(S) \right). \qquad (2)$$

Mathematically, while the Shapley value is the weighted average of marginal contributions of players to coalitions, the Banzhaf value is a simple average.

Unfortunately, the difference between these two values when applied to feature attribution has not been understood well in the literature. We note that explanations based on Shapley value have been extensively studied experimentally [Lundberg et al., 2020, Lundberg and Lee, 2017, Štrumbelj and Kononenko, 2014, Sundararajan et al., 2017], whereas in the case of Banzhaf value, such studies have been done only on some basic datasets [Datta et al., 2015, Sliwinski et al., 2019, Patel et al., 2020]. Moreover, despite very high similarity of both methods, to the best of our knowledge, no comparison between them has been done on real-world data-sets, e.g., Patel et al. [2020] compares on a single depth-3 tree, whereas Patel et al. [2021] uses both methods for vocabulary selection in different NLP tasks without directly comparing these methods. For completeness, we review other explanation methods in Appendix C.

Our contribution. In this paper we fill this gap by providing a comprehensive analysis of the Banzhaf value, including its comparison to the Shapley value, when applied to explainability of tree ensemble models. In particular, our contributions can be summarised as follows. We first show that, for tree ensemble models, when using the same natural set function g as in [Lundberg and Lee, 2017, Lundberg et al., 2018, 2020], Banzhaf value can be computed in linear time, noticeably faster than the Shapley value. Specifically, we develop an $O(TL + n)$ time algorithm for computing the Banzhaf value-based explanation of a tree ensemble model's output. Here, T is the number of trees, L is the maximum number of leaves in a tree, and n is the number of features. In comparison, the state-of-the-art Shapley value-based algorithm by Lundberg et al. [2018, 2020] runs in $O(TLD^2 + n)$ time, where D denotes the maximum depth of a tree in the ensemble. We note that recent papers [Arenas et al., 2021, Van den Broeck et al., 2021] do not improve this complexity[1], but extend the method to more complex models instead.[2] We stress that our algorithm is *asymptotically optimal*, since even the description of a tree ensemble has size $O(TL)$, and the output has size $O(n)$.

On the technical level, the algorithm of Lundberg et al. [2018, 2020], reduces computing $(\phi_i)_{i=1}^n$ to finding individual *leaf contributions* to the explanation, one per each leaf/feature pair (l, i) such that i is used as a split feature in some ancestor of l. This goal is achieved using a top-down recursive algorithm whose running time is inherently $\Omega(TLD)$ (i.e., super-linear in the input size) simply because there can be $\Omega(TLD)$ such leaf/feature pairs. This bound still holds even when this approach is applied to computing a Banzhaf-value explanation. In our approach, leaf contributions are aggregated using a more efficient bottom-up dynamic programming approach, which requires only a *linear* number of auxiliary values to be computed.

In the experiments, our algorithm visibly outperforms all other algorithms, and can lead to considerable time savings when computing feature importances for decision tree-based models in practice. Moreover, we analytically prove that for trees of depths that commonly occur in practice, our algorithm for the Banzhaf value delivers numerically correct results. Similar arguments do not seem to be applicable to the most efficient algorithms computing Shapley value based explanations even for constant depth trees.

We also perform an experimental comparison of the Banzhaf and Shapley values for tree ensemble models. For four studied real-world datasets and using two different approaches to training tree models, we verify experimentally that both methods deliver essentially the same average feature importance scores (called *global impacts* in [Lundberg et al., 2020]) and very close explanations of individual predictions. However, the Banzhaf value is more numerically robust than the Shapley value, and only very small errors are observed in the computations. Overall, our analysis indicates that for tree models, the Banzhaf value has two important advantages over the Shapley value. While both methods deliver comparable explanations, the Banzhaf value works faster and is less prone to numerical errors.

2 PRELIMINARIES

Let $U := \{1, \ldots, n\}$ be a set of *features*. Let x be the input to the model to be explained. For $i \in U$, we write x_i to refer to the *value* of the i-th feature in x. More generally, for any subset $S \subseteq U$ we write x_S when referring to the vector $(x_i)_{i \in S}$. We sometimes talk about random feature vectors, or consider the values of individual features to be random variables. We then write X or X_i respectively. We write X_S to denote the vector of random variables $(X_i)_{i \in S}$. Let $\bar{S}$ denote the complement $U \setminus S$.

Tree models. Let $f : \mathbb{R}^U \to \mathbb{R}$ be the output function of the model to be explained. We focus on tree ensemble models $(\mathcal{T})_{i=1}^T$ where the output $f(x)$ of the model is simply the

[1] In fact, these papers only focus on proving polynomial time complexity, and neither bound nor optimize the degrees of the actual polynomials involved. Obtaining low-degree polynomial time algorithms is crucial from the practical point of view.

[2] Though, not always without loss of generality wrt. to [Lundberg et al., 2018, 2020]. For example, decision trees captured by the class of boolean circuits studied in [Arenas et al., 2021] seem to forbid using a single feature for splitting multiple times on a root-leaf path of a decision tree.

Algorithm 1 Estimating $\mathbb{E}[f(x_S, X_{\bar{S}})]$

function $\texttt{Desc}(S, v)$
  if v is a leaf **then**
    return $f(v)$
  else
    if $d_v \in S$ **then**
      if $x_{d_v} < t_v$ **then**
        return $\texttt{Desc}(S, a_v)$
      else
        return $\texttt{Desc}(S, b_v)$
    else
      return $\frac{1}{r_v} \cdot (r_{a_v} \cdot \texttt{Desc}(S, a_v) + r_{b_v} \cdot \texttt{Desc}(S, b_v))$
function $g(S)$
  return $\frac{1}{T} \cdot \sum_{i=1}^{T} \texttt{Desc}(S, \rho_i)$

average output $f_{\mathcal{T}_i}(x)$ of its T individual trees. Following Lundberg et al. [2020], we assume the individual trees to have the number of leaves bounded by L and depth bounded by D.[3] Let us denote by ρ_i the root of the tree $\mathcal{T}_i$.

When talking about an input decision tree $\mathcal{T}$, we adopt the notation of [Lundberg et al., 2020]. $\mathcal{T}$ is a binary tree based on single-variable splits: each non-leaf node $v \in \mathcal{T}$ is assigned a *feature* d_v, and a *threshold* t_v, whereas each leaf l is assigned rea *value* $f(l)$. Let a_v, b_v denote the left and right children of a non-leaf node $v \in \mathcal{T}$. The output $f_{\mathcal{T}}(x)$ of the tree $\mathcal{T}$ is computed by following a root-leaf path in $\mathcal{T}$: at a non-leaf node $v \in \mathcal{T}$, we descend to a_v if $x_{d_v} < t_v$, or to b_v otherwise. When a leaf is reached, its value is returned. Denote by $\mathcal{L}(\mathcal{T})$ the set of leaves of $\mathcal{T}$. Denote by $\mathcal{T}[v]$ the subtree of $\mathcal{T}$ rooted at v.

Set functions. We write $f(x_S, X_{\bar{S}})$ when referring to a random variable being the value of f if the values for features in S are fixed to the respective values of x, and the values $X_{\bar{S}}$ are random variables. Let X_U be distributed as in the training set of the model f.

Recall that a *set function* $g : 2^U \to \mathbb{R}$ with $g(\emptyset) = 0$, has to be fixed to talk about the Shapley or Banzhaf value-based explanations $(\phi_i)_{i \in U}$ and $(\beta_i)_{i \in U}$ as defined in Equations (1) and (2), resp., Lundberg et al. [2020] and Janzing et al. [2020] suggest using the following idealized set function g^* for feature attribution:

$$g^*(S) := \mathbb{E}[f(x_S, X_{\bar{S}})] - \mathbb{E}[f(X_U)]. \tag{3}$$

Note how the term $\mathbb{E}[f(X_U)]$ in (3) cancels out when computing the Shapley value from Equation (1). Thus, for simplicity in the following we can redefine $g^*(S) := \mathbb{E}[f(x_S, X_{\bar{S}})]$.

[3]This is merely for clarity of the obtained time bounds. See discussion after Theorem 1.

Using the idealized set function g^* would be computationally too costly. Consequently, Lundberg et al. [2020] in their TREESHAP_PATH[4] algorithm considers the set function g whose purpose is to "approximate" g^*. Namely, $g(S) \approx g^*(S)$ is computed as shown in Algorithm 1. This method dates back to the classical work of Friedman [2001] and is also implemented as a way to compute partial dependence plots in the scikit-learn package [Pedregosa et al., 2011]. Its one advantage is that it does not require access to the training data, but merely to the "coverages" r_v of all the subtrees $\mathcal{T}[v]$ (for all trees $\mathcal{T}$ in the ensemble), i.e., the numbers of training set points that fall into $\mathcal{T}[v]$. It can be proved that this method approximates $\mathbb{E}[f(x_S, X_{\bar{S}})]$ well if the individual feature random variables X_i are independent. With such a set function g, Lundberg et al. [2018, 2020] show how to compute the Shapley value explanations $(\phi_i)_{i \in U}$ exactly in $O(TLD^2 + n)$ time.

In the remaining part of the paper, we denote by $g(S)$ the output of Algorithm 1 for the subset $S \subseteq U$, i.e., we consider the same approximation of $g^*(S)$ as in the TREESHAP_PATH algorithm of Lundberg et al. [2020].

3 THE BANZHAF VALUE ALGORITHM

In this section, we introduce an optimal $O(TL + n)$ time algorithm, called BANZHAF, for computing explanations based on the Banzhaf value. For clarity, let us assume first that there is just a single tree $\mathcal{T}$ in the model, i.e., $T = 1$. This is without loss of generality, since the prediction of an ensemble model is simply the average of the predictions produced by individual trees. We will describe the algorithm for arbitrary T later on. Due to space constraints, the proofs of technical lemmas can be found in Appendix D.

Let ρ denote the root of $\mathcal{T}$, and p_v the parent of node $v \in \mathcal{T}$, $v \neq \rho$. Furthermore, let F_v be the set of features assigned to the ancestors of v, i.e., $F_\rho = \emptyset$, and $F_v = F_{p_v} \cup \{d_{p_v}\}$ for $v \neq \rho$. The value $P[v] = r_v / r_\rho$ can be thought as the probability that the model returns a value from $\mathcal{T}[v]$.

Algorithm 1 computes the estimate $\mathbb{E}[f(x_S, X_{\bar{S}})]$. Observe that the output of this algorithm for $S = \emptyset$ is precisely equal to $\sum_{l \in \mathcal{L}(\mathcal{T})} P[l] \cdot f(l)$. More generally, denote by $P[v, S]$ the weight from the ancestor recursive calls assigned to the subtree rooted at v when running Algorithm 1 with an arbitrary $S \subseteq U$. Formally, $P[\rho, S] = 1$, and for any $v \neq \rho$,

$$P[v, S] = \begin{cases} P[p_v, S] \cdot \frac{r_v}{r_{p_v}} & \text{if } d_{p_v} \notin S, \\ P[p_v, S] \cdot [x_{d_{p_v}} < t_{p_v}] & \text{if } d_{p_v} \in S, v = a_{p_v}, \\ P[p_v, S] \cdot [x_{d_{p_v}} \geq t_{p_v}] & \text{if } d_{p_v} \in S, v = b_{p_v}. \end{cases}$$

Then, the algorithm outputs

$$\sum_{l \in \mathcal{L}(\mathcal{T})} P[l, S] \cdot f(l) = g(S) \approx g^*(S). \tag{4}$$

[4]We will sometimes use an abbreviated name TREESHAP.

In our approach, each of the desired attributions β_i is obtained by summing the contributions of each individual leaf $l \in \mathcal{L}(\mathcal{T})$ with $i \in F_l$ to the sum (2) with g defined as in (4). More precisely:

$$\beta_i = \sum_{l \in \mathcal{L}(\mathcal{T})} \left(\frac{f(l)}{2^{n-1}} \sum_{S \subseteq U \setminus \{i\}} \left(P[l, S \cup \{i\}] - P[l, S] \right) \right).$$

We now introduce the following crucial intermediate values that will enable us to evaluate the above formula efficiently. For any $v \in \mathcal{T}$, and subset $G \subseteq U$, let

$$\beta(v, G) := \frac{1}{2^{|G|}} \sum_{S \subseteq G} P[v, S].$$

Lemma 1. *For any $i \in U$, we have:*

$$\beta_i = \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} 2f(l) \cdot \left(\beta(l, F_l) - \beta(l, F_l \setminus \{i\}) \right).$$

Lemma 1 reduces computing the Banzhaf value to computing $O(L)$ values of the form $\beta(l, F_l)$, and $O(L \cdot D)$ values of the form $\beta(l, F_l \setminus \{i\})$, for all (l, i) such that $l \in \mathcal{L}(\mathcal{T})$ and $i \in F_l$. This $O(L \cdot D)$ bound follows since each leaf has no more than D ancestors, which implies $|F_l| \leq D$.

In the following part of the section, we first give a recursive formula for computing the values $\beta(v, G)$ efficiently using dynamic programming. Next, we show a simpler $O(LD)$ time algorithm computing all the values $\beta(\cdot, \cdot)$ required by Lemma 1. As a final step, we show how to improve the worst-case running time of the algorithm to optimal $O(L)$.

Recurrence. To proceed, we will need the auxiliary values $\Delta_{v,y}$ for $v \in \mathcal{T}$ and $y \in U$, defined inductively as follows:

$$\Delta_{v,y} = \begin{cases} 1 & \text{if } v = \rho, \\ \Delta_{p_v,y} & \text{if } d_{p_v} \neq y \text{ and } v \neq \rho, \\ \Delta_{p_v,y} \cdot [x_y < t_{p_v}] \cdot \frac{r_v}{r_{p_v}} & \text{if } d_{p_v} = y \text{ and } a_{p_v} = v \neq \rho, \\ \Delta_{p_v,y} \cdot [x_y \geq t_{p_v}] \cdot \frac{r_v}{r_{p_v}} & \text{if } d_{p_v} = y \text{ and } b_{p_v} = v \neq \rho. \end{cases}$$

The above auxiliary values can be in turn used to recursively compute the values $P[\cdot, \cdot]$.

Lemma 2. *Let $v \in \mathcal{T}$ and $Q \subseteq U$ and $y \in U \setminus Q$. Then:*

$$P[v, Q \cup \{y\}] = P[v, Q] \cdot \Delta_{v,y}.$$

The values $\beta(v, G)$ can be computed recursively, as stated in the below lemmas.

Lemma 3. *Let $v \in \mathcal{T}$ and $G \subseteq U$. Let $y \in U \setminus G$. Then:*

$$\beta(v, G \cup \{y\}) = \frac{1}{2} \left(1 + \Delta_{v,y}\right) \beta(v, G).$$

Lemma 4. *Let $v \in \mathcal{T}$, $v \neq \rho$. Then, for any $Q \subseteq U \setminus \{d_{p_v}\}$,*

$$\beta(v, Q) = \beta(p_v, Q) \cdot \frac{r_v}{r_{p_v}}.$$

Algorithm 2 Computing $\beta[l] = \beta(l, F_l)$ for all $l \in \mathcal{L}(\mathcal{T})$.

procedure TRAVERSE(v)
  if $d_{p_v} \in F$ **then**
    present := **true** ▷ record that d_{p_v} in F_{p_v}
    $b := \frac{2}{1+\delta[d_{p_v}]} \cdot \beta[p_v]$ ▷ $b = \beta(p_v, F_{p_v} \setminus d_{p_v})$
  else
    present := **false**
    $F := F \cup \{d_{p_v}\}$ ▷ ensure $F = F_v$
    $b := \beta[p_v]$ ▷ $b = \beta(p_v, F_{p_v} \setminus d_{p_v})$
  $\delta_{\text{old}} := \delta[d_{p_v}]$
  if $v = a_{p_v}$ **then**
    $\delta[d_{p_v}] := \delta[d_{p_v}] \cdot [x_y < t_{p_v}] \cdot \frac{r_v}{r_{p_v}}$
  else
    $\delta[d_{p_v}] := \delta[d_{p_v}] \cdot [x_y \geq t_{p_v}] \cdot \frac{r_v}{r_{p_v}}$
  $\delta^*[v] := \delta[d_{p_v}]$ ▷ store $\Delta_{v,d_{p_v}}$ for future use
  $b := b \cdot r_v / r_{p_v}$ ▷ $b = \beta(p_v, F_v)$
  $\beta[v] := b \cdot \frac{1}{2}(1 + \delta[d_{p_v}])$ ▷ Lemma 3
  if $v \notin \mathcal{L}(\mathcal{T})$ **then**
    TRAVERSE(a_v)
    TRAVERSE(b_v)
  if present = **false then** ▷ revert changes to F, δ
    $F := F \setminus d_{p_v}$
  $\delta[d_{p_v}] := \delta_{\text{old}}$

3.1 BASIC ALGORITHM

Equipped with Lemmas 3 and 4, one can easily move between "nearby" values $\beta(G, v)$. Namely, for any $i \in U$, given $\beta(v, G)$ and $\Delta_{v,i}$, each of the values $\beta(a_v, G)$, $\beta(b_v, G)$, $\beta(v, G \cup \{i\})$ can be computed in $O(1)$ time.

Moreover, the values $\beta(p_v, G)$, $\beta(v, G \setminus \{i\})$ can also be obtained in $O(1)$ time by applying the respective "inverse" forms of these lemmas. We now stress that being able to compute $\beta(v, G \setminus \{i\})$ out of a state of the form $\beta(v, G)$, i.e., removing elements from the feature set G, is crucial for two reasons. First, recall that we need to obtain values of the form $\beta(l, F_l \setminus \{i\})$ for all leaves l and all $i \in F_l$. For all such i, this state can be obtained using a single inverse application of Lemma 3. Moreover, applying Lemma 4 to obtain a state $\beta(v, F_v)$ out of the parent state $\beta(p_v, F_{p_v})$ requires $d_{p_v} \notin F_{p_v}$. This may be violated if $F_v = F_{p_v}$, i.e., d_{p_v} is a feature in some other ancestors of v in the tree (which does happen in practical models). In such a case, the inverse Lemma 3 can be used to first compute $\beta(p_v, F_v \setminus \{d_{p_v}\})$, then apply Lemma 4 to obtain $\beta(v, F_v \setminus \{d_{p_v}\})$, and finally again use Lemma 3 to get $\beta(v, F_v)$.

The basic algorithm (which is similar in its essence to TREESHAP_PATH), computes all the values $\beta(v, F_v)$ for $v \in \mathcal{T}$ – as explained above – using a simple recursive tree traversal in $O(L)$ time. In particular, this also gives all the values $\beta(l, F_l)$ that we need when invoking Lemma 1. Afterwards, for each leaf $l \in \mathcal{T}$, the remaining (again, re-

quired by the formula in Lemma 1) $|F_l|$ values of the form $\beta(l, F_l \setminus \{i\})$ for $i \in F_l$ can be computed in $O(1)$ extra time each using Lemma 3. As a result, through all pairs (l, i), this takes $O\left(\sum_{l \in \mathcal{L}(\mathcal{T})} |F_l|\right) = O(LD)$ time.

The above analysis silently assumed that all the needed auxiliary values $\Delta_{v,y}$ can be accessed in $O(1)$ time. We now justify this assumption. During the tree traversal we store a global array δ indexed with the features U. We maintain an invariant that $\delta[y]$ equals $\Delta_{p_v,y}$ when the processing of a vertex v starts and also when it finishes. By (3), to guarantee the invariant is satisfied upon the recursive traversals of the subtrees rooted at a_v or b_v, we may possibly need to update only the value $\delta[d_v]$ according to (3), because $\Delta_{v,y} \neq \Delta_{a_v,y}$ or $\Delta_{v,y} \neq \Delta_{b_v,y}$ may only happen when $y = d_v$. When a recursive traversal returns, we revert that change to $\delta[d_v]$.

The pseudocode of a recursive procedure TRAVERSE computing all the values $\beta(l, F_l)$, which we also require in our optimal algorithm, is given as Algorithm 2. In this procedure, each of the computed values $\beta(v, F_v)$ is recorded in a global array as $\beta[v]$. The auxiliary global variable F stores the set F_v when node v is processed; F can be implemented using a bitmap of size n.

3.2 THE OPTIMAL ALGORITHM

The high-level idea behind our improved algorithm is to avoid computing all the leaf contributions to the individual components β_i of the Banzhaf value separately. Instead, for every node $v \in \mathcal{T}$, $v \neq \rho$, such that $d_{p_v} = i$, we compute the total contribution to β_i of *all* the leaves $\mathcal{L}_v \subseteq \mathcal{T}[v]$, defined to be the subset of leaves for which v constitutes the *nearest* weak ancestor (i.e., a node is considered its own ancestor) with $d_{p_v} = i$, at once.

Note that for a given $i \in U$, the sets $\mathcal{L}_v$ for $v \in \mathcal{T}$ satisfying $d_{p_v} = i$, are pairwise disjoint, and in fact form a partition of the set $\{l \in \mathcal{L}(\mathcal{T}) : i \in F_l\}$ through which summation in Lemma 1 is performed. Additionally, observe that the values $\Delta_{l,d_{p_v}}$ are equal to $\Delta_{v,d_{p_v}}$ for all leaves l in $\mathcal{L}_v$.

Consider the following values for all $v \in \mathcal{T}$, $v \neq \rho$:

$$B(v) = \sum_{l \in \mathcal{L}_v} f(l) \cdot \beta(l, F_l).$$

The below lemma shows that computing the Banzhaf value β can be reduced, in linear time, to computing all the values $B(v)$, $v \in \mathcal{T} \setminus \{\rho\}$: indeed, each $B(v)$ appears in the sum below for precisely one $i \in U$.

Lemma 5. *For any $i \in U$, we have:*

$$\beta_i = \sum_{\substack{v \in \mathcal{T} \setminus \{\rho\} \\ d_{p_v} = i}} \frac{2(\Delta_{v,i} - 1)}{1 + \Delta_{v,i}} \cdot B(v).$$

We have previously showed that the values $\beta(l, F_l)$ can be computed in linear time. We now describe a recursive procedure FAST(u), where $u \neq \rho$, computing $B(v)$ for all $v \in \mathcal{T}[u]$ in a bottom-up manner. Let

Algorithm 3 Computing the values $B(v)$ for all $v \in \mathcal{T}$.

```
procedure FAST(v)
    H[d_{p_v}].PUSH(v)
    if v ∈ L(T) then
        S[v] := f(v) · β[v]
    else
        FAST(a_v)
        FAST(b_v)
        S[v] := S[a_v] + S[b_v]
    z := 0                    ▷ z stores the sum Σ_{w∈Q_v} S(w)
    while H[d_{p_v}].TOP() ≠ v do
        z := z + S[H[d_{p_v}].TOP()]
        H[d_{p_v}].POP()
    B[v] := S[v] − z
    if |H[d_{p_v}]| = 1 then      ▷ empty H[d_{p_v}] if g_v = ⊥
        H[d_{p_v}].POP()
```

$$S(v) = \sum_{v \in \mathcal{L}(\mathcal{T}[v])} f(l) \cdot \beta(l, F_l),$$

that is, $S(v)$ sums the values $f(l) \cdot \beta(l, F_l)$ in $\mathcal{T}[v]$. Clearly, for each $l \in \mathcal{L}(\mathcal{T})$, we have $S(l) = f(l) \cdot \beta(l, F_l)$, and for a non-leaf $v \in \mathcal{T}$, $S(v) = S(a_v) + S(b_v)$ holds. As a result, all the values $S(v)$ can be computed in linear time using a bottom-up computation over the tree.

Given the sums $S(v)$, we proceed as follows. For $v \in \mathcal{T}$, let Q_v be the set of non-leaf nodes $w \in \mathcal{T}[v]$ with $d_{p_w} = d_{p_v}$ and v is the nearest ancestor of w with $d_{p_w} = d_{p_v}$. We have: $\mathcal{L}_v = \mathcal{L}(\mathcal{T}[v]) \setminus \left(\bigcup_{w \in Q_v} \mathcal{L}(\mathcal{T}[w])\right)$, and thus

$$B(v) = S(v) - \sum_{w \in Q_v} S(w).$$

Observe that the total size of sets Q_v (over all $v \in \mathcal{T}$) is $O(L)$, so if we are allowed to iterate through Q_v whenever we wish to compute $B(v)$, the computation of $B(v)$ takes $O(L)$ time as well. We now explain how to accomplish this. Let g_w denote the nearest ancestor of $w \in \mathcal{T}$ with $d_{p_w} = d_{p_{g_w}}$. One way to enable iterating through Q_v when v is processed bottom-up, is to maintain, for each feature $j \in U$, a global stack $H[j]$ containing all the nodes w such that $d_{p_w} = j$ and that the computation for w (i.e., the call FAST(w)) has already been started or completed, but the computation for g_w has not yet completed. The stack elements are sorted using the pre-order of the nodes of v, so that the node w with the highest pre-order is at the top of $H[d_{p_w}]$. The stack can be updated in $O(1)$ time whenever a recursive call starts. Observe that $v \in H[d_{p_v}]$ when FAST(v) has started but has not yet finished. Now, given $H[d_{p_v}]$, it is enough to note that Q_v equals precisely the set of elements of $H[d_{p_v}]$ closer to the top of the stack than v. Thus, one

Algorithm 4 Computing the explanations $(\beta_i)_{i=1}^n$ of the tree ensemble model's $(\mathcal{T}_j)_{j=1}^T$ prediction $f(x)$.

function BANZHAFEXPLAIN$(n, (\mathcal{T}_j)_{j=1}^T)$
  for $i \in U$ **do** ▷ initialize global data
    $\beta_i := \beta[i] := 0$ ▷ $(\beta_i)_{i=1}^n$ stores the result
    $\delta[i] := 1$
    $H[i] =$ empty stack
  $F := \emptyset$
  for $j = 1, \ldots, T$ **do**
    $\rho :=$ the root node of $\mathcal{T}_j$
    for $v \in \{a_\rho, b_\rho\}$ **do**
      TRAVERSE(v)
      FAST(v)
    for $v \in \mathcal{T}_j \setminus \{\rho\}$ **do**
      $\beta_{d_v} := \beta_{d_v} + \frac{2(\delta^*[v]-1)}{1+\delta^*[v]} \cdot B[v]$ ▷ Lemma 5
  return $(\beta_i/T)_{i=1}^n$ ▷ average through the T trees

can indeed iterate through Q_v in $O(|Q_v|)$ time as desired. Moreover, Q_v constitutes precisely the set of elements that have to be popped from $H[d_{p_v}]$ when FAST(v) returns. The asymptotic cost of popping stack elements can charged to the corresponding pushes and thus can be neglected.

A pseudocode of the procedure FAST computing all the values $B(v)$ given the values $\beta(l, F_l)$ is given in Algorithm 3. In Algorithm 4 we give a pseudocode of the full algorithm computing the Banzhaf value-based explanations for a tree ensemble model $(\mathcal{T}_j)_{j=1}^T$. Since the value of such a model is defined to be the average prediction over all the individual tree predictions, the final explanation is simply the average of the individual explanations. We have thus proved:

Theorem 1. *Let $n = |U|$. The Banzhaf value-based explanation $(\beta)_{i \in U}$ of a prediction of a tree ensemble model consisting of T trees with at most L leaves each, can be computed in optimal $O(TL + n)$ time.*

We remark that if the ensemble contains T trees of very different sizes, the time can be more precisely bounded by $O\left(\sum_{i=1}^T |\mathcal{T}_i| + n\right)$, i.e., remains optimal in the input size.

Finally, it is worth noting that the above approach to speeding-up the basic algorithm can be also successfully applied to reduce the time complexity of the `TREESHAP_PATH` explanation algorithm of Lundberg et al. [2020] from $O(TLD^2 + n)$ to $O(TLD + n)$. Due to space constraints, we defer the details to Appendix E.

4 EXPERIMENTAL ANALYSIS

The goals of our experiments are threefold:

- *Time performance* — first, we test the performance of the `BANZHAF` algorithm proposed in the previous section and compare it to the performance of the `TREESHAP_PATH` algorithm by Lundberg et al. [2020]—the state-of-the-art algorithm for the Shapley value explanations for tree models.
- *Qualitative differences* — next, we investigate whether the Banzhaf value returns qualitatively different results than the Shapley value for tree models.
- *Numerical accuracy* — finally, we compare numerical accuracy of both algorithms.

Name	Rows	Cols	tree depth	iter.	max depth	learning rate
`BOSTON`	506	13	10	100	6	0.01
`NHANES`	8023	79	40	250	4	0.2
`VEH.INS.`	304887	14	60	250	4	0.2
`FLIGHTS`	1543718	647	100	250	10	0.2

Table 1: The sizes of datasets and parametrisation of the experiments. The "tree depth" column reports tree_depth of the decision tree (DT) with all the other parameters set to default values. The "iterations", "max depth" and "learning rate" columns are the parameters used for training xgboost.

4.1 EXPERIMENTAL SETUP AND DATASETS

In our experiment we use both the sklearn implementation of Decision Trees (DT) or xgboost implementation of Gradient Boosting Decision Trees (GBDT). These are some of the most popular algorithms for generating decision trees and are quite often used for large depths of trees. Using large-depth trees is particularly beneficial for datasets with many features and complex relationship between them (see e.g., [Bordag et al., 2021, Pham et al., 2019] for a usage of trees of depth 100). Let us emphasize that large depth of a tree, e.g. depth 100, does not mean the size of the tree is 2^{100}, because trees might be (and usually are) unbalanced. To simplify the experiments and reduce the their running times, we trained the DT algorithm to generate only one tree. We use four "real-world" datasets (see Table 1 for key details):

1. `BOSTON` (abbr. `BS`). [bos]. This small prediction dataset contains information concerning housing in the area of Boston Massachusetts. The task is to predict the price of the house.
2. `NHANES` (`NH`). The same dataset that was used in previous work on tree model interpretability [Lundberg et al., 2020] which our work most closely relates to. The parameters used for training were the same as in [Lundberg et al., 2020].
3. `VEHICLE_INSURANCE` (`VI`). [veh]. A medium size dataset for predicting who might be interested in vehicle insurance based on health insurance data.
4. `FLIGHTS` (`FL`). [fli]. A large dataset for predicting the flights' delays. A large number of columns was caused by one-hot encoding 'UniqueCarrier', 'Origin', 'Dest',

	BANZHAF	TREESHAP		BANZHAF	TREESHAP
BS_GB	0.48 s	0.70 s	BS_DT	0.41 s	0.41 s
VI_GB	23.63 s	35.32 s	NH_DT	3.57 s	42.87 s
NH_GB	50.20 s	1 m 28 s	VI_DT	4 m 55 s	30 m 55 s
FL_GB	13 m 18 s	48 m 8 s	FL_DT	14 m 28 s	5 h 9 m

Table 2: Running times of the two explanation algorithms on the entire dataset. We observe that BANZHAF is substantially faster than TREESHAP_PATH on each instance.

'CancellationCode' in a standard way, i.e., for each possible value v of a given column c we created additional categorical column c_v ($v \in \{0, 1\}$) indicating that the value of c equals v iff the value of c_v equals 1.

We will refer to the above datasets by adding "DT" and "GB" suffixes (for DT and GBDT algorithms, resp.) to the ordinal name of the prediction dataset. Note that the parameters were not extensively tuned since our main goal here centers around interpreting models and not optimizing them.

All our experiments were performed using Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz with 512 Gb of RAM using only one thread for computation. The operating system was Ubuntu 18.04.2 LTS. Our linear-time BANZHAF algorithm was implemented in C++, whereas for TREESHAP_PATH, we used to original C language implementation from the SHAP package [sha]. The binaries were compiled using clang version 6.0.0-1ubuntu2 with -O3 optimization.

4.2 COMPARISON OF RUNNING TIMES

In this section, we compare the running times of the algorithms. For each of the instances, the task was to compute the explanations of *all* individual data points. In Table 2 we show the running times for different examples. We conclude that BANZHAF is consistently faster than TREESHAP_PATH, and using it can lead to considerable time savings for larger data-sets. As anticipated by the theoretical worst-case time complexity analysis, the observed speed-up increases with the depth of trees in the model.

4.3 COMPARISON OF FEATURE SCORES

We test whether the Banzhaf value assigns qualitatively different importance to features than the Shapley value. The comparison is performed from two viewpoints.

Global importance. First, we compare the global importances of individual features for the model. To this end, we apply the same measure of *global impact* of a feature as in [Lundberg et al., 2020]. Let $\mathcal{D}$ be some dataset. Suppose for each $i \in U$ we have some feature attribution function $\gamma_i : \mathcal{D} \to \mathbb{R}$. Let us consider the global impact of the feature over dataset $\mathcal{D}$ measured as $\Gamma_i = \sum_{x \in \mathcal{D}} |\gamma_i(x)|$. For example, we can set $\gamma_i = \phi_i$ to get a *Shapley global impact* Φ_i, or $\gamma_i = \beta_i$ to get a *Banzhaf global impact* B_i.

For each of the datasets and algorithms we computed and plotted the Shapley and Banzhaf global impacts. The obtained plots can be found in Appendix A.

For NHANES, BOSTON, and VEHICLE_INSURANCE datasets, the obtained plots of Banzhaf/Shapley global impacts, computed using BANZHAF and TREESHAP_PATH respectively, are virtually indistinguishable. For the larger instance based on the dataset FLIGHTS, only very small differences in the ordering of features by importance can be observed for both FLIGHTS_GB and FLIGHTS_DT.

Specific data points. We now turn to describing how much the obtained Banzhaf and Shapley explanations deviate from each other for specific data points. To measure the difference between the feature orderings produced by both methods, we computed the *modified Cayley distance* between the respective orderings of $n \in \{3, 10, 20\}$ most important features for each data point, and took the average over all data points. The Cayley distance measures the number of swaps needed to switch from one permutation to another. In our modified version, we also support the case where the sets of considered most important features in the respective permutations are different. For a missing feature, we add it at the end of the permutation. The results are presented in Table 3. They confirm that the differences are on average small; in particular for the instances BOSTON_GB, NHANES_GB, and VEHICLE_INSURANCE_GB, for 98% of the data points, the respective 3 top features and their order matched. The orderings deviation was generally larger for DT instances where larger tree depths were allowed.

We also studied per-feature average differences between the values of Banzhaf and Shapley explanations for each of the datasets. See Appendix B for the relevant plots. We consider both MAE (Mean Average Error) and RMSE (Root Mean Square Error). Formally, for each dataset $\mathcal{D}$ out of those and each feature i used therein, these are defined as: $\text{MAE}_i = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} |\phi_i(x) - \beta_i(x)|$ and $\text{RMSE}_i = \sqrt{\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} (\phi_i(x) - \beta_i(x))^2}$. Here, $\phi_i(x)$ denotes the Shapley explanation of $f(x)$ for data point $x \in \mathcal{D}$, as computed by TREESHAP_PATH. Similarly, $\beta_i(x)$ denotes the Banzhaf explanation as computed by BANZHAF.

For the "smaller" instances BOSTON_GB, NHANES_GB, and VEHICLE_INSURANCE_GB and all features, the observed MAE and RMSE errors did not exceed 5% of the corresponding global impacts. For the remaining larger models, the MAE error did not exceed 20% for the top features. On the other hand, for the large-depth FLIGHTS_DT model, the RMSE error reached around 50% even for the top features, which suggests there were data points with very big absolute differences in the produced explanations. These errors indicate that when looking at specific data points one should expect only small differences in the ordering of features and only for features with similar scores. The differences are expected to be larger for larger models.

Ins/n	3	10	20	Ins/n	3	10	20
BOS_GB	0.02	1.05		BOS_DT	0.08	1.7	
NH_GB	0.01	0.34	1.53	NH_DT	0.29	3.69	10.79
VI_GB	0.02	0.73		VI_DT	0.13	2.60	
FL_GB	0.4	3.08	8.63	FL_DT	0.18	3.38	10.59

Table 3: The average modified Cayley distance for the n most important features for $n \in \{3, 10, 20\}$ produced by BANZHAF and TREESHAP_PATH algorithms.

4.4 NUMERICAL ACCURACY

The fact that the more significant differences between the obtained importances arised for large models suggested that the compared explanation algorithms might suffer numerical problems. To investigate this possibility and compare numerical stability of BANZHAF and TREESHAP_PATH, we considered a simple artificially prepared instance SYNTHETIC_SPARSE for which we know the answer for both the Shapley value and the Banzhaf value.

In the SYNTHETIC_SPARSE instance, the set of features is $U = \{1, \ldots, d\}$, where d is a depth parameter. The instance contains one tree and one data point $x = [1, \ldots, 1] \in \mathbb{R}^d$. The tree consists of two subtrees of the same shape and depth $d - 1$. All values $f(l)$ in the leaves are equal to 0 and 777 in the left and the right subtree of the root, resp. All leaves l have coverages equal to 33. Every internal node of depth i has one leaf child, and one non-leaf child, whose (inductively defined) subtree has depth $i - 1$. The split condition in an internal node at depth i is $x_{d-i} < 1$. In this instance, the only feature with a nonzero Shapley/Banzhaf importance, equal to 388.5, is the feature d used to split at depth 0. All other features have importances equal to 0.[5]

We have observed that for trees of depth $d \approx 50$, errors dominate the results, i.e., the relative error exceeds 1. In Figure 1 we visualise the mean absolute errors for TREESHAP_PATH and BANZHAF for the SYNTHETIC_SPARSE instance.

We now give a potential reason why the Banzhaf value-based implementations may be much more stable in terms of the produced relative errors. Recall that the values $\beta(l, F_l)$ for all $l \in \mathcal{L}(\mathcal{T})$, $i \in F_l$ are computed via dynamic programming using Lemmas 3 and 4. Hence, these values are all computed via multiplications and divisions on *positive* numbers lying roughly between 0.5 and r_ρ. In fact, the intermediate values $\beta(v, F_v)$ can be obtained via $O(1)$ applications of Lemmas 3 and 4 from the "parent" value $\beta(p_v, F_{p_v})$. Such a computation can be proven to introduce a multiplicative error between $1/(1+\epsilon)^{O(1)}$ and $(1+\epsilon)^{O(1)}$, where ϵ is the machine epsilon. This in turn implies a relative error bound of $(1+\epsilon)^{O(1)} - 1$. Moreover, by induction on the tree depth, we can easily obtain (see Appendix F for a formal proof):

[5]This follows by the *sensitivity* axiom (see, e.g., [Janzing et al., 2020]) that both Banzhaf and Shapley values satisfy.

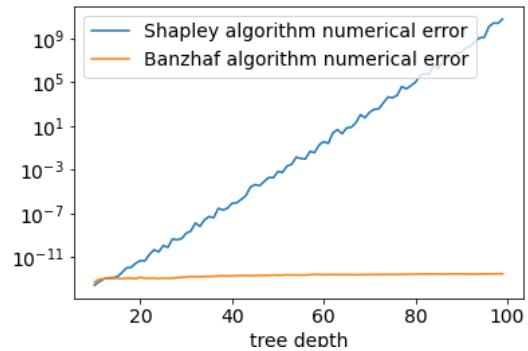


Figure 1: The numerical error for SYNTHETIC_SPARSE.

Lemma 6. *The leaf values $\beta(l, F_l)$ can be computed with relative error at most $(1+\epsilon)^{O(D)} - 1$.*

This bound is quite pessimistic and at the same time not very large if double precision is used and the tree depth D is small enough. On the other hand, if one considers computing the Shapley value explanations, if one wants to retain the $O(LD^2)$ time bound of the TREESHAP_PATH algorithm [Lundberg et al., 2020], then it seems that *subtractions* of intermediate values are inherent. Roughly speaking, this is because for Shapley-based explanations, if one applies an analogous dynamic programming approach, then the Shapley-analogue of Lemma 3 involves a recursive formula that is a *sum* of two "earlier" dynamic programming cells.[6] Recall, however, that our (and also Lundberg et al.'s) approach also required *inverse* applications of Lemma 3, especially when a single feature may appear multiple times on a root-leaf path. For Shapley value such an inverse application involves subtraction of equally-signed numbers.[7]

It is unclear if a similar (to Lemma 6) relative error bound can be proven in presence of such subtractions, which in general may lead to so-called *catastrophic cancellations*.

5 CONCLUSIONS

The contribution of this paper is twofold. First, we have developed an efficient algorithm for computing feature importance measures for tree ensemble models that is based on the Banzhaf value. This result improves the running time of previous state of the art. Second, we present the first extensive comparison between the Shapley and Banzhaf values in this context. We observe that both methods deliver explanations of essentially the same strength by returning almost the same ordering of features. However, these experimental results indicate that the Banzhaf value has an important advantage over the Shapley value, i.e., it allows for faster algorithms as well as these algorithms make much lower numerical errors.

[6]See Lemma 7 in Appendix E.

[7]In the original TREESHAP algorithm subtractions of this kind manifest in line 31 of [Lundberg et al., 2019, Algorithm 2].

References

BOSTON dataset. URL https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html.

FLIGHTS dataset. URL https://www.kaggle.com/abdurrehmankhalid/delayedflights.

SHAP python package. URL https://github.com/slundberg/shap.

VEHICLE INSURANCE dataset. URL https://www.kaggle.com/anmolkumar/health-insurance-cross-sell-prediction?select=train.csv.

M. Ancona, Enea Ceolini, C. Öztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *ICLR*, 2018.

Marcelo Arenas, Pablo Barceló, Leopoldo Bertossi, and Mikaël Monet. The tractability of shap-score-based explanations for classification over deterministic and decomposable boolean circuits. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6670–6678, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16825.

S. Bach, Alexander Binder, Grégoire Montavon, F. Klauschen, K. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10, 2015.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(61):1803–1831, 2010.

Natalie Bordag, Elmar Zügner, Pablo López-García, Selina Kofler, Martina Tomberger, Abdullah Al-Baghdadi, Jessica Schweiger, Yasemin Erdem, Christoph Magnes, Saiki Hidekazu, Wolfgang Wadsak, Björn-Thoralf Erxleben, and Barbara Prietl. Towards fast, routine blood sample quality evaluation by probe electrospray ionization (pesi) metabolomics. *medRxiv*, 2021. doi: 10.1101/2021.04.18.21254782. URL https://www.medrxiv.org/content/early/2021/04/23/2021.04.18.21254782.

L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2004.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and Regression Trees*. CRC Press, 1984.

S. Chebrolu, A. Abraham, and J. Thomas. Feature deduction and ensemble design of intrusion detection systems. *Comput. Secur.*, 24:295–307, 2005.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

A. Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. *IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016.

Amit Datta, Anupam Datta, Ariel D. Procaccia, and Yair Zick. Influence in classification via cooperative game theory. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 511–517. AAAI Press, 2015. ISBN 9781577357384.

Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19(2):257–266, 1994.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

Nicholas J. Higham. *Accuracy and stability of numerical algorithms, Second Edition*. SIAM, 2002. ISBN 978-0-89871-521-7. doi: 10.1137/1.9780898718027. URL https://doi.org/10.1137/1.9780898718027.

V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts. Inferring regulatory networks from expression data using tree-based methods. *PLoS ONE*, 5, 2010.

Dominik Janzing, Lenon Minorics, and Patrick Blöbaum. Feature relevance quantification in explainable AI: A causal problem. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS*, volume 108, pages 2907–2916. PMLR, 2020.

Kaggle. 2017 kaggle machine learning data science survey, 2017. URL https://www.kaggle.com/kaggle/kaggle-survey-2017.

P. Kindermans, Kristof T. Schütt, M. Alber, K. Müller, D. Erhan, Been Kim, and S. Dähne. Learning how to explain neural networks: Patternnet and patternattribution. In *ICLR*, 2018.

S. Lipovetsky and M. Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17:319–330, 2001.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774. Curran Associates, Inc., 2017.

Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Scott M. Lundberg, Gabriel G. Erion, Hugh Chen, Alex J. DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. Explainable AI for trees: From local explanations to global understanding. *CoRR*, abs/1905.04610, 2019. URL http://arxiv.org/abs/1905.04610.

Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, Jan 2020.

Khaled Maafa, Lhouari Nourine, and Mohammed Said Radjef. Algorithms for computing the shapley value of cooperative games on lattices. *Discrete Applied Mathematics*, 249:91–105, 2018.

Tomasz P Michalak, Karthik V Aadithya, Piotr L Szczepanski, Balaraman Ravindran, and Nicholas R Jennings. Efficient computation of the shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research*, 46:607–650, 2013.

Neel Patel, Martin Strobel, and Yair Zick. High dimensional model explanations: an axiomatic approach, 2020.

Roma Patel, Marta Garnelo, Ian M. Gemp, Chris Dyer, and Yoram Bachrach. Game-theoretic vocabulary selection via the shapley value and banzhaf index. In *NAACL*, 2021.

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*, 12:2825–2830, 2011.

Hung N. Pham, T. T. Do, Kelvin Yi Jie Chan, Gopa Sen, Andy Han, Pier Lim, Teresa Siew Loon Cheng, Quang H. Nguyen, Binh P. Nguyen, and Matthew C. H. Chua. Multimodal detection of parkinson disease based on vocal and improved spiral test. *2019 International Conference on System Science and Engineering (ICSSE)*, pages 279–284, 2019.

Gregory Plumb, Denali Molitor, and Ameet S Talwalkar. Model agnostic supervised local explanations. In *Advances in Neural Information Processing Systems*, volume 31, pages 2515–2524. Curran Associates, Inc., 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

A. Roth. The shapley value : essays in honor of Lloyd S. Shapley. *Cambridge University Press*, 1988.

A. Saabas. Treeinterpreter python package. URL https://github.com/andosa/treeinterpreter.

M. Sandri and P. Zuccolotto. A bias correction algorithm for the gini variable importance measure in classification trees. *Journal of Computational and Graphical Statistics*, 17:611 – 628, 2008.

L. S. Shapley. A Value for n-Person Games. *Contributions to the Theory of Games 2.28*, pages 307–317, 1953.

A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *ArXiv*, abs/1605.01713, 2016.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, volume 70, pages 3145–3153. PMLR, 2017.

Jakub Sliwinski, Martin Strobel, and Yair Zick. Axiomatic characterization of data-driven influence measures for classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):718–725, Jul. 2019.

Jost Tobias Springenberg, A. Dosovitskiy, T. Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2015.

Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665, 2014. doi: 10.1007/s10115-013-0679-x.

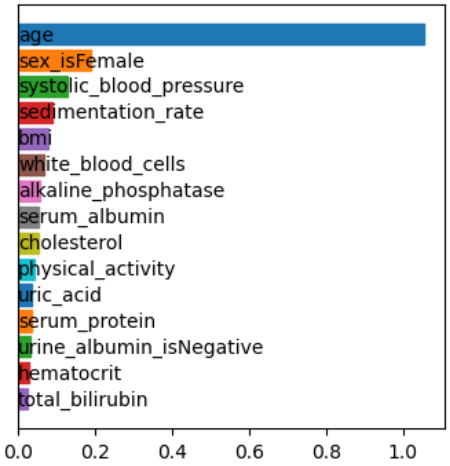
Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *Proceedings of the 37th International Conference on Machine Learning ICML*, volume 119, pages 9269–9278. PMLR, 2020.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, volume 70, pages 3319–3328. PMLR, 2017.

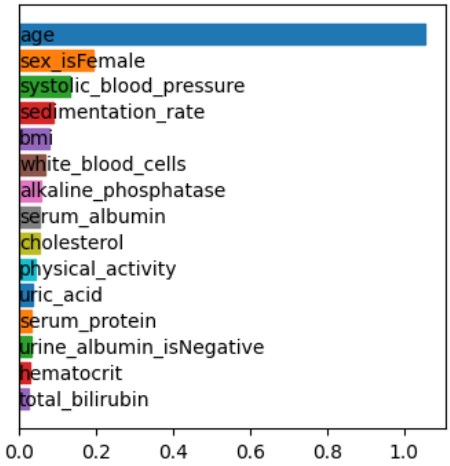
Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of shap explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6505–6513, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16806.

Matthew D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

A GLOBAL IMPACTS COMPARISON

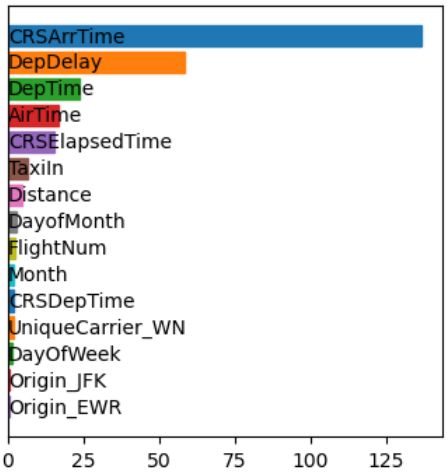


(a) Global Shapley impact obtained with TREESHAP_PATH.

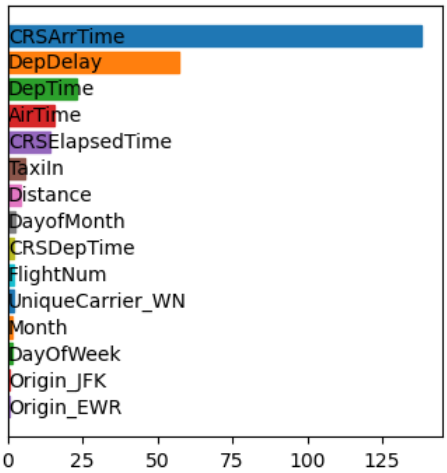


(b) Global Banzhaf impact obtained with BANZHAF.

Figure 2: The global impacts of individual features for the NHANES_GB dataset.

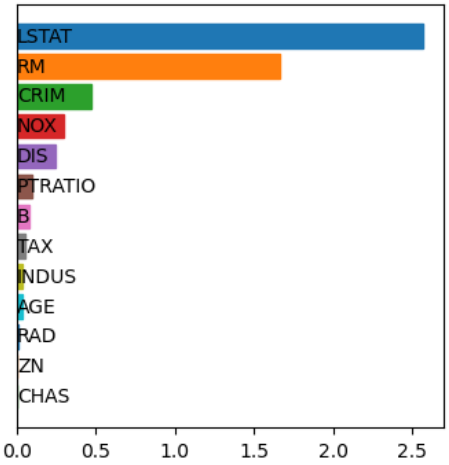


(a) Global Shapley impact obtained with TREESHAP_PATH.

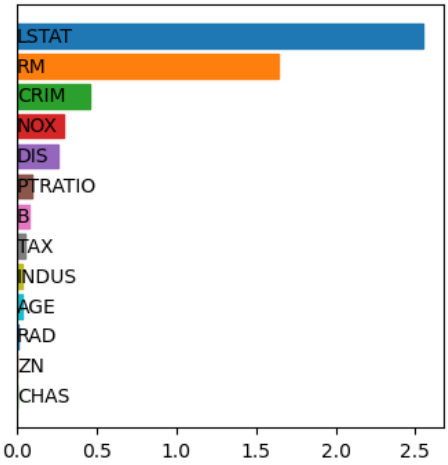


(b) Global Banzhaf impact obtained with BANZHAF.

Figure 3: The global impacts of individual features for the FLIGHTS_DT dataset. We observe small differences in the ordering of less important features.

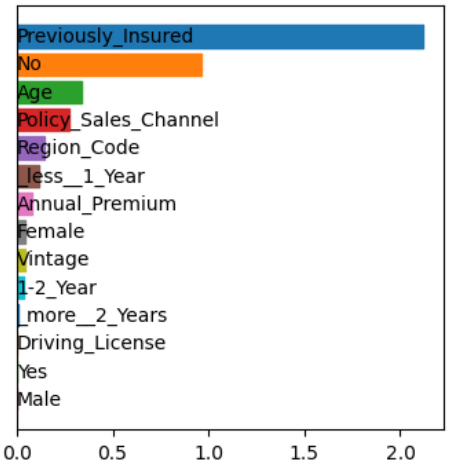


(a) The original Shapley value.

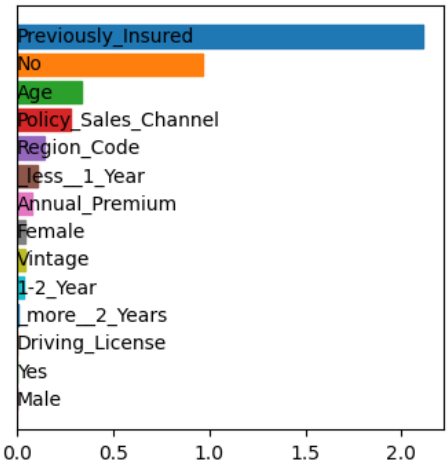


(b) The Banzhaf value.

Figure 4: The global impacts of the individual features for the BOSTON_GB dataset. We observe that the plots are indistinguishable.

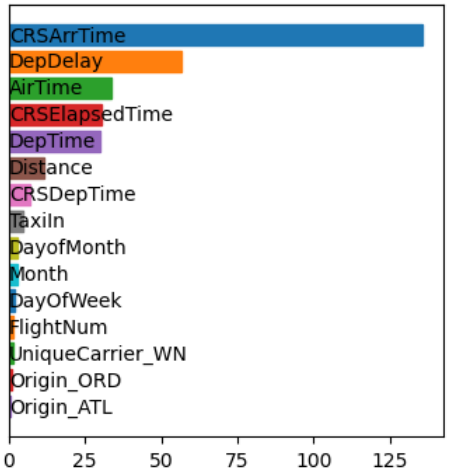


(a) The original Shapley value.

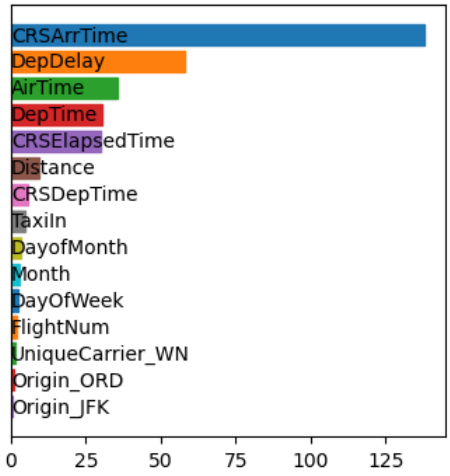


(b) The Banzhaf value.

Figure 5: The features' global impacts for the VEHICLE_INSURANCE_GB dataset. We observe that the plots are indistinguishable.

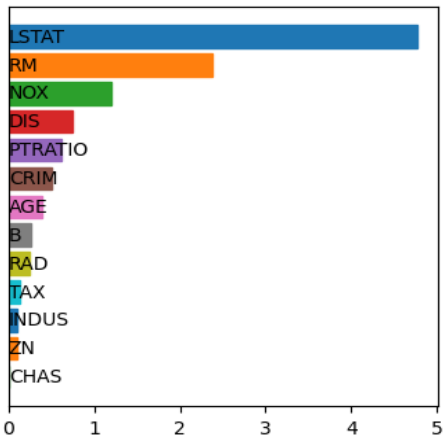


(a) The original Shapley value.

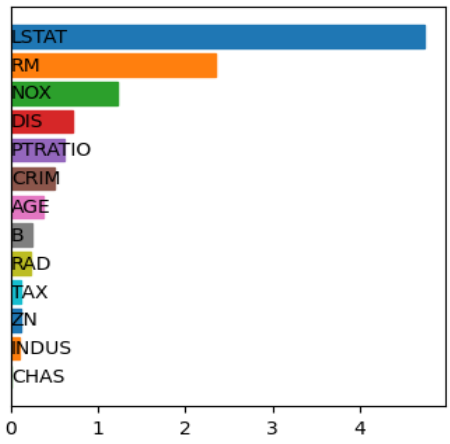


(b) The Banzhaf value.

Figure 6: The global impacts of the individual features for the `FLIGHTS_GB` dataset. We observe small differences in the ordering.

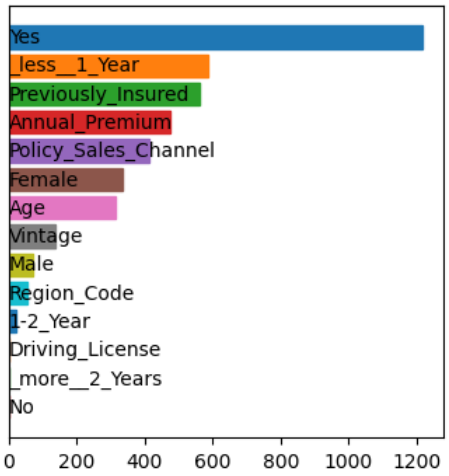


(a) The original Shapley value.

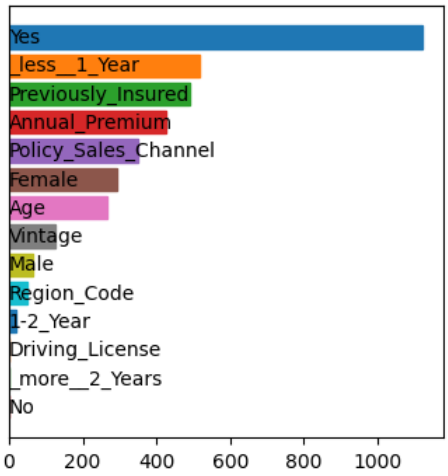


(b) The Banzhaf value.

Figure 7: The global impacts of the individual features for the `BOSTON_DT` dataset. We observe minor differences between plots.

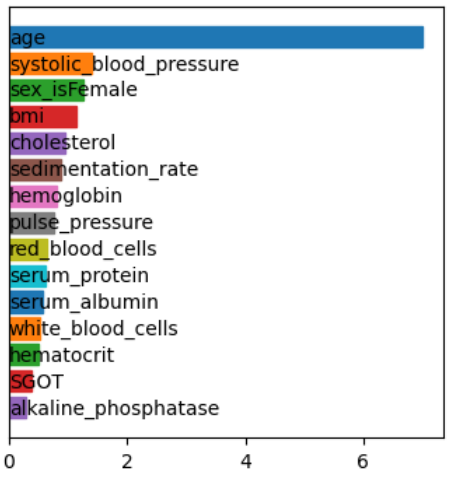


(a) The original Shapley value.

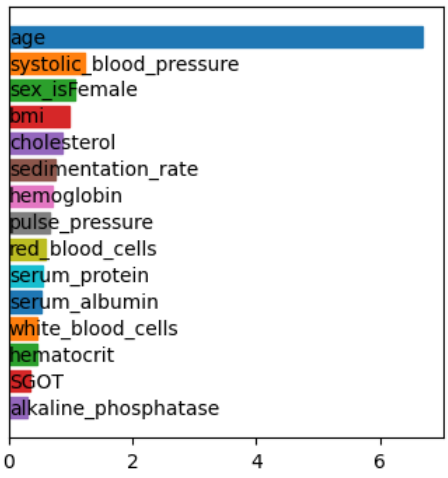


(b) The Banzhaf value.

Figure 8: The features' global impacts for the `VEHICLE_INSURANCE_DT` dataset. The plots are almost indentical.



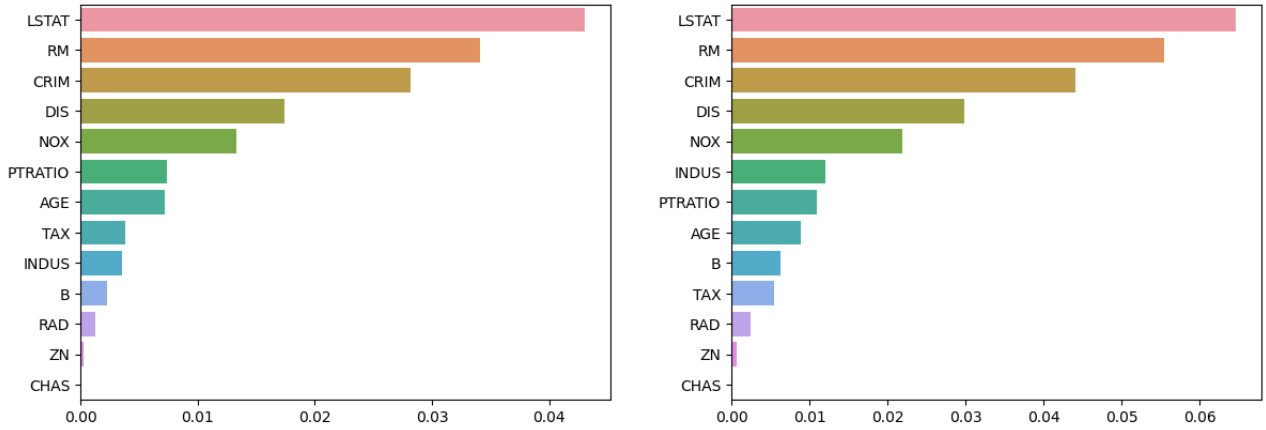
(a) The original Shapley value.



(b) The Banzhaf value.

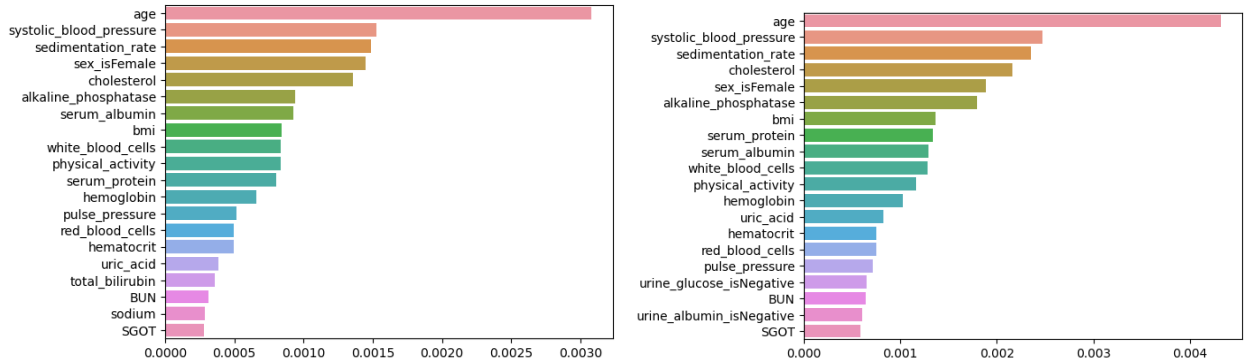
Figure 9: The global impacts of the individual features for the `NHANES_DT` dataset. The plots are almost identical.

B PER-FEATURE MAE AND RMSE ERRORS



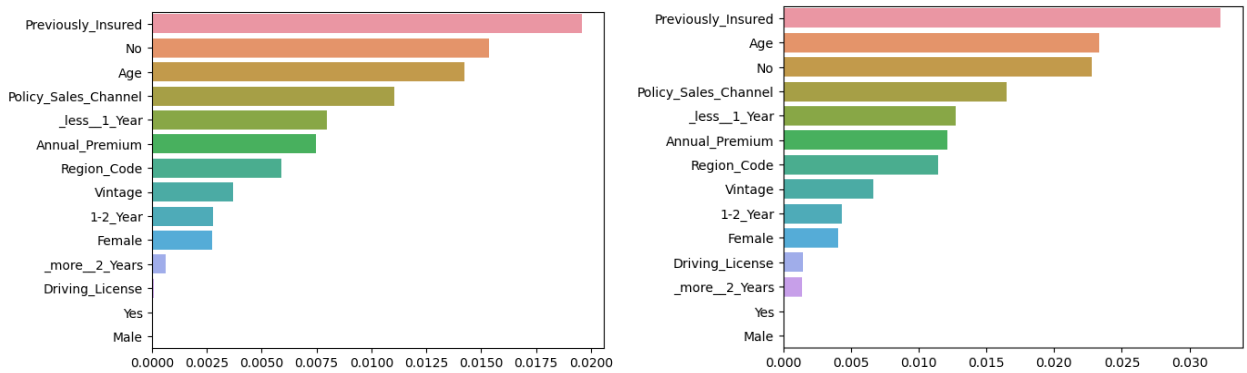
(a) MAE error. (b) RMSE error.

Figure 10: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `BOSTON_GB` dataset.



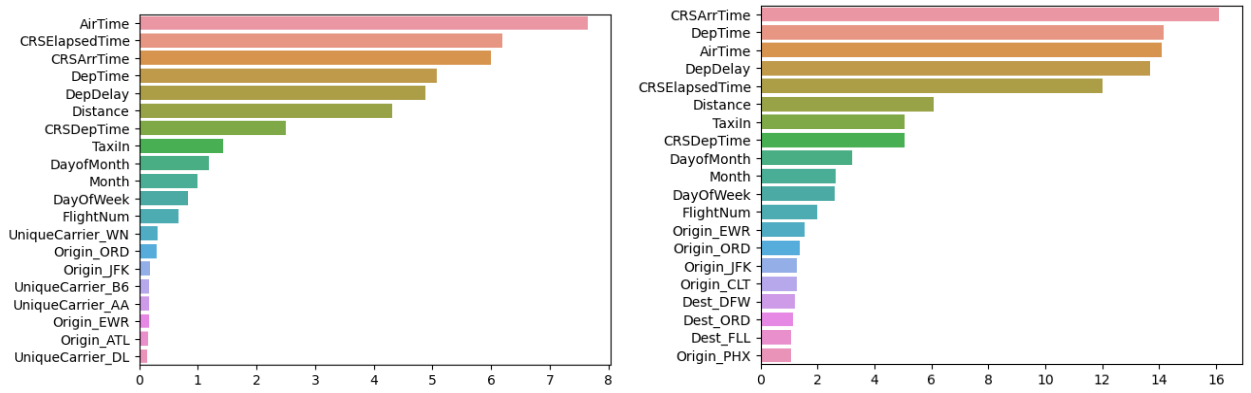
(a) MAE error. (b) RMSE error.

Figure 11: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `NHANES_GB` dataset.



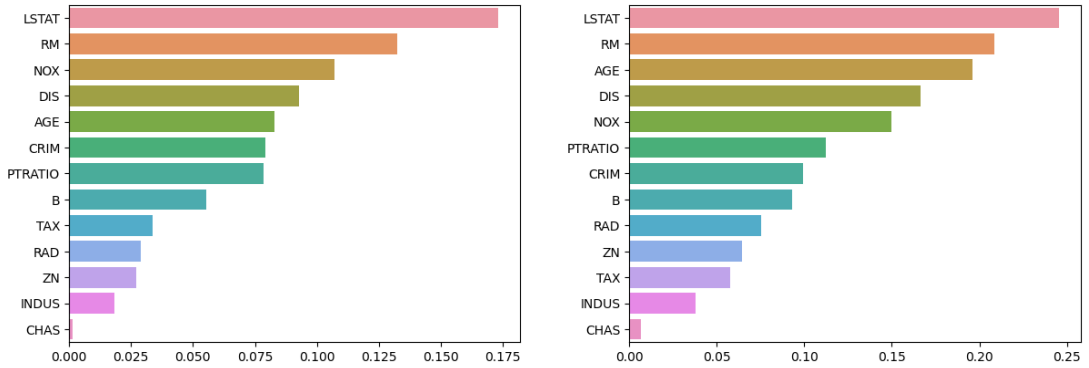
(a) MAE error. (b) RMSE error.

Figure 12: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `VEHICLE_INSURANCE_GB` dataset.



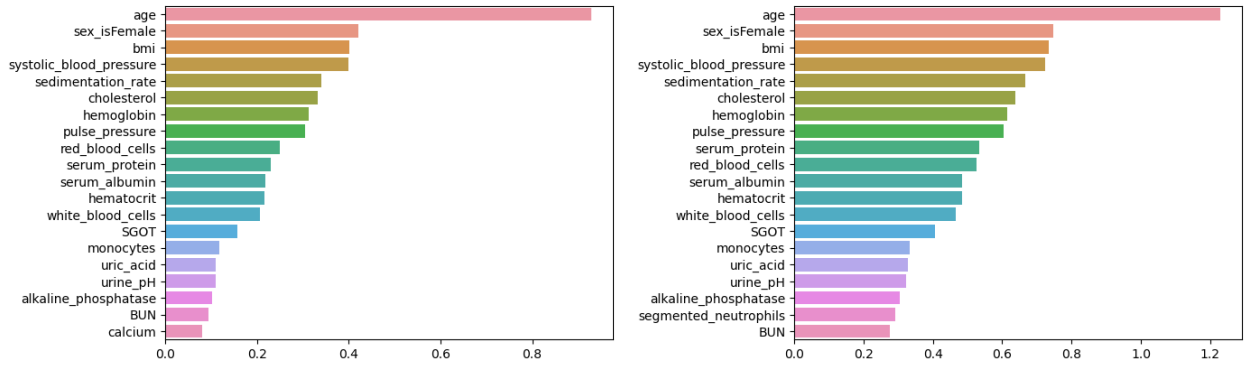
(a) MAE error. (b) RMSE error.

Figure 13: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `FLIGHTS_GB` dataset.



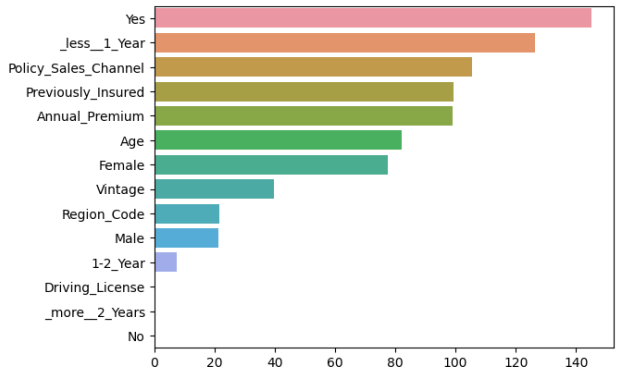
(a) MAE error. (b) RMSE error.

Figure 14: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `BOSTON_DT` dataset.

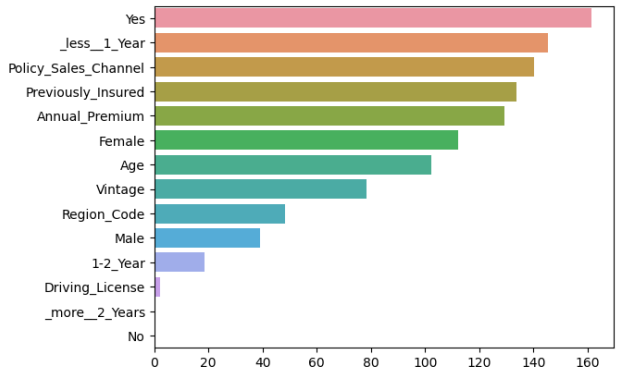


(a) MAE error. (b) RMSE error.

Figure 15: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `NHANES_DT` dataset.

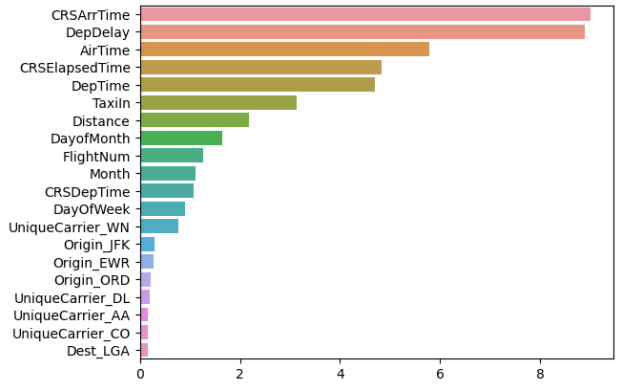


(a) MAE error.

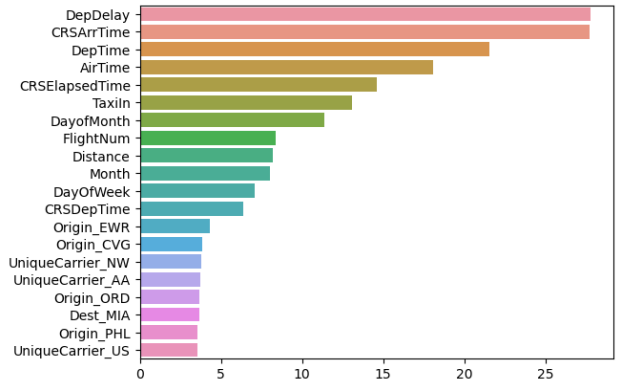


(b) RMSE error.

Figure 16: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `VEHICLE_INSURANCE_DT` dataset.



(a) MAE error.



(b) RMSE error.

Figure 17: The MAE and RMSE error between the Banzhaf value and the Shapley value for the `FLIGHTS_DT` dataset.

C FURTHER RELATED WORK

Feature importance values summarize a complicated ensemble model and provide insight into what features drive the model's prediction. There can be various types of explanation methods to compute such values: model-dependent or model-agnostic methods, global or local explanation methods.

Explanation methods for trees. Global feature importance values are computed for an entire dataset in mainly three different ways. The basic global approach, *Split Count*, is to count the number of times a feature is used for splitting [Chen and Guestrin, 2016]. However, this method fails to account for the impacts of different splits. The *Gain* approach to feature importance [Breiman et al., 1984] is to attribute the reduction of loss contributed by each split in each decision tree and it is widely used as the basis for feature selection methods [Chebrolu et al., 2005, Huynh-Thu et al., 2010, Sandri and Zuccolotto, 2008]. Another commonly used approach, *Permutation*, is to randomly permute the data column corresponding to a feature in the test set and observe the change in the model's loss [Breiman, 2004]. If the model is heavily dependent on the feature then permuting it should create a large increase in the model's loss. These approaches are designed to estimate the global importance of a feature over an entire dataset, so they are not directly applicable to local explanations that are specific to each prediction. Local explanation methods for computing feature importance values for a single prediction are not well studied for trees. Only a couple of tree-specific local explanation methods were known previously. One is to just report the decision path, which is not useful for large tree ensembles. The other one is by Saabas which is a heuristic method that measures the difference in the model's expected output. The Saabas method explains a prediction by following the decision path of the current input and attributing the differences in the expected output of the model to each of the features along the path. The expected value of every node in the tree is the average of the model output over the training samples going through that node. For explaining an ensemble model made of many trees, the Saabas value for the ensemble is defined as the sum of the values for each tree. As noted in [Lundberg et al., 2018], the feature importance values from the gain, split count, and Saabas methods are all inconsistent i.e., a model can be modified so that it relies more on a given feature, yet the importance assigned to that feature decreases.

Model-agnostic methods. One of the most common local explanation methods in deep learning literature is to take the gradient of the model's output with respect to its inputs at the current sample or multiplying the gradient times the value of the input features. As depending entirely on the gradient of the model at a single point can often be misleading [Shrikumar et al., 2016] various other methods have also been proposed [Springenberg et al., 2015, Zeiler and Fergus, 2014, Bach et al., 2015, Shrikumar et al., 2016, Kindermans et al., 2018, Ancona et al., 2018]. Model-agnostic methods on the other hand make no assumptions about the internal structure of the model and depend on the relationship between changes in the model inputs and model outputs. This is achieved by training a global mimic model to approximate the original model, then locally explaining the mimic model [Baehrens et al., 2010, Plumb et al., 2018]. Alternatively, the mimic model can be fit into the original model locally for each prediction. In the LIME method [Ribeiro et al., 2016] the coefficients are used as an explanation for a local linear mimic model. In [Ribeiro et al., 2018] the rules are used as the explanation for a local decision rule mimic model. Recently, several methods for the local explanation of model predictions (such as LIME [Ribeiro et al., 2016], DeepLIFT [Shrikumar et al., 2016, 2017], Layer-wise Relevance Propagation [Bach et al., 2015], and three methods from cooperative game theory: Shapley regression values [Lipovetsky and Conklin, 2001], Shapley sampling values [Štrumbelj and Kononenko, 2014], and Quantitative Input Influence [Datta et al., 2016]) are unified into a single class of *additive feature attribution methods* [Lundberg and Lee, 2017]. This class contains methods that explain a model's output as a sum of real values attributed to each input feature. It is of particular interest as there is a unique optimal explanation approach in the class that satisfies three desirable properties: local accuracy, missingness, and consistency [Roth, 1988, Shapley, 1953]. *Local accuracy* (also called *Efficiency* or *Completeness*) means that the sum of the feature attributions is equal to the output of the function we want to explain. *Missingness* (also called *Sensitivity*, or *Null-player axiom*) means that missing features are given no importance and *Consistency* (also called *Monotonicity*) means that if a feature has a larger impact on the model after a change then the attribution assigned to that feature can only increase. One can use model-agnostic local explanation methods to explain tree models however their dependence on post-hoc modeling of an arbitrary function can make them slow or might suffer from sampling variability for models with many input features [Lundberg et al., 2020]. Although such methods are often practical for individual explanations, but can quickly become impractical for explaining entire datasets.

D OMITTED PROOFS

Lemma 2. *Let $v \in \mathcal{T}$ and $Q \subseteq U$ and $y \in U \setminus Q$. Then:*

$$P[v, Q \cup \{y\}] = P[v, Q] \cdot \Delta_{v,y}.$$

Proof. The proof proceeds by induction on the depth of v in $\mathcal{T}$. The claim holds obviously for $v = \rho$. So suppose v is non-root.

Assume first that $d_{p_v} \notin Q \cup \{y\}$. Then, by applying the definition of $P[\cdot, \cdot]$ twice, and the induction hypothesis:

$$\begin{aligned} P[v, Q \cup \{y\}] &= P[p_v, Q \cup \{y\}] \cdot \frac{r_v}{r_{p_v}} \\ &= P[p_v, Q] \cdot \Delta_{p_v,y} \cdot \frac{r_v}{r_{p_v}} \\ &= P[v, Q] \cdot \frac{r_{p_v}}{r_v} \cdot \Delta_{p_v,y} \cdot \frac{r_v}{r_{p_v}} \\ &= P[v, Q] \cdot \Delta_{v,y}. \end{aligned}$$

Otherwise, $d_{p_v} \in Q \cup \{y\}$. Assume wlog. $v = a_{p_v}$ – the case $v = b_{p_v}$ is symmetric. We have:

$$\begin{aligned} P[v, Q \cup \{y\}] &= P[p_v, Q \cup \{y\}] \cdot [x_{d_{p_v}} < t_{p_v}] \\ &= P[p_v, Q] \cdot \Delta_{p_v,y} \cdot [x_{d_{p_v}} < t_{p_v}] \end{aligned}$$

If $d_{p_v} = y$, then $d_{p_v} \notin Q$ and we have $\Delta_{v,y} = \Delta_{p_v,y} \cdot [x_y < t_{p_v}] \cdot \frac{r_v}{r_{p_v}}$. So in that case

$$P[v, Q \cup \{y\}] = P[p_v, Q] \cdot \Delta_{v,y} \cdot \frac{r_{p_v}}{r_v} = P[v, Q] \cdot \Delta_{v,y}.$$

If, on the other hand, we have $y \neq d_{p_v} \in Q$, then:

$$P[v, Q \cup \{y\}] = P[p_v, Q] \cdot \Delta_{p_v,y} \cdot [x_{d_{p_v}} < t_{p_v}] = P[v, Q] \cdot \Delta_{v,y}.$$ □

Lemma 3. *Let $v \in \mathcal{T}$ and $G \subseteq U$. Let $y \in U \setminus G$. Then:*

$$\beta(v, G \cup \{y\}) = \frac{1}{2}\left(1 + \Delta_{v,y}\right)\beta(v, G).$$

Proof. Let $m = |G|$. By the definition and Lemma 2, we have:

$$\begin{aligned} \beta(v, G \cup \{y\}) &= \sum_{S \subseteq G \cup \{y\}} \frac{1}{2^{m+1}} P[v, S] \\ &= \sum_{S \subseteq G} \frac{1}{2^{m+1}} P[v, S] + \sum_{y \in S \subseteq G \cup \{y\}} \frac{1}{2^{m+1}} P[v, S] \\ &= \sum_{S \subseteq G} \frac{1}{2} \cdot \frac{1}{2^m} P[v, S] + \sum_{S \subseteq G} \frac{1}{2} \cdot \Delta_{v,y} \cdot \frac{1}{2^m} P[v, S] \\ &= \frac{1}{2}\left(1 + \Delta_{v,y}\right)\beta(v, G). \end{aligned}$$ □

Lemma 4. *Let $v \in \mathcal{T}$, $v \neq \rho$. Then, for any $Q \subseteq U \setminus \{d_{p_v}\}$,*

$$\beta(v, Q) = \beta(p_v, Q) \cdot \frac{r_v}{r_{p_v}}.$$

Proof. The claim follows easily by the definition of $\beta(v, Q)$ and since $P[v, G] = P[p_v, G] \cdot \frac{r_v}{r_{p_v}}$ holds for every subset $G \subseteq Q$. □

Lemma 1. *For any $i \in U$, we have:*

$$\beta_i = \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} 2f(l) \cdot (\beta(l, F_l) - \beta(l, F_l \setminus \{i\})).$$

Proof. By Lemmas 2 and 3, we have:

$$\begin{aligned}
\beta_i &= \sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot \left(\frac{1}{2^{n-1}} \sum_{S \subseteq U \setminus \{i\}} (P[l, S \cup \{i\}] - P[l, S]) \right) \\
&= \sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot \left(\left(\frac{\Delta_{l,i}}{2^{n-1}} \sum_{S \subseteq U \setminus \{i\}} P[l, S] \right) - \beta(l, U \setminus \{i\}) \right) \\
&= \sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot (\Delta_{l,i} \cdot \beta(l, U \setminus \{i\}) - \beta(l, U \setminus \{i\})) \\
&= \sum_{l \in \mathcal{L}(\mathcal{T})} 2f(l) \cdot (\beta(l, U) - \beta(l, U \setminus \{i\})).
\end{aligned}$$

Note that if $y \notin F_l$, then $\Delta_{l,y} = 1$, and thus by Lemma 3, we have $\beta(l, X \cup \{y\}) = \beta(l, X)$ for any $X \subseteq U \setminus \{y\}$. Inductively we obtain $\beta(l, X \cup Y) = \beta(l, X)$ for any $Y \subseteq U \setminus X \setminus F_l$. In particular, we obtain $\beta(l, U) = \beta(l, F_l)$, and $\beta(l, U \setminus \{i\}) = \beta(l, F_l \setminus \{i\})$. To finish the proof, observe that if $i \notin F_l$, then $\beta(l, F_l) = \beta(l, F_l \setminus \{i\})$ by Lemma 3, so for such i the summand above will be equal to 0. □

Lemma 5. *For any $i \in U$, we have:*

$$\beta_i = \sum_{\substack{v \in \mathcal{T} \setminus \{\rho\} \\ d_{p_v} = i}} \frac{2(\Delta_{v,i} - 1)}{1 + \Delta_{v,i}} \cdot B(v).$$

Proof. Recall from the proof of Lemma 1 that

$$\beta_i = \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} f(l) \cdot (\Delta_{l,i} - 1) \cdot \beta(l, F_l \setminus \{i\}).$$

By changing the order of summation, we equivalently have:

$$\begin{aligned}
\beta_i &= \sum_{\substack{v \in \mathcal{T} \\ d_{p_v} = i}} \sum_{l \in \mathcal{L}_v} f(l) \cdot (\Delta_{l,i} - 1) \cdot \beta(l, F_l \setminus \{d_{p_v}\}) \\
&= \sum_{\substack{v \in \mathcal{T} \\ d_{p_v} = i}} \sum_{l \in \mathcal{L}_v} f(l) \cdot (\Delta_{l,i} - 1) \cdot \frac{2}{1 + \Delta_{l,i}} \cdot \beta(l, F_l).
\end{aligned}$$

and the lemma follows by the definition of $B(v)$ and $\Delta_{l,i} = \Delta_{v,i}$. □

E IMPROVED ALGORITHM FOR SHAPLEY EXPLANATIONS

In this section we sketch the changes that need to be made to the algorithms of Section 3 to make it compute Shapley value-based explanations as given by (1).

We use intermediate values $\phi(\cdot, \cdot, \cdot)$ analogous to the values $\beta(\cdot, \cdot)$ that constituted the base of the Banzhaf algorithm. For any vertex $v \in \mathcal{T}$, set $G \subseteq U$ and integer $k = 0, \ldots, |G|$, let

$$\phi(v, G, k) := \frac{1}{|G| + 1} \sum_{\substack{S \subseteq G \\ |S| = k}} \binom{|G|}{k}^{-1} \cdot P[v, S]. \tag{5}$$

Let us also put $\phi(v, G)$ to be a vector consisting of all the values $\phi(v, G, \cdot)$:

$$\phi(v, G) := (\phi(v, G, k))_{k=0}^{|G|}.$$

We have the following analogues of Lemma 3 and Lemma 4, respectively. For convenience, let us define $\phi(v, G, k) = 0$ for $k < 0$ or $k > |G|$.

Lemma 7. *Let $v \in \mathcal{T}$, $G \subseteq U$ and $k \in \{0, \dots, |G|\}$. Let $y \in U \setminus G$. Then:*

$$\phi(v, G \cup \{y\}, k) = \frac{|G| + 1 - k}{|G| + 2} \cdot \phi(v, G, k) + \frac{k}{|G| + 2} \cdot \Delta_{v,y} \cdot \phi(v, G, k - 1).$$

Proof. Let $m = |G| + 1$. By Lemma 2 we get:

$$\begin{aligned}
\phi(v, G \cup \{y\}, k) &= \sum_{\substack{S \subseteq G \cup \{y\} \\ |S| = k}} \frac{1}{m+1} \binom{m}{k}^{-1} P[v, S] \\
&= \left(\sum_{\substack{S \subseteq G \\ |S| = k}} \frac{1}{m+1} \binom{m}{k}^{-1} P[v, S] \right) + \left(\sum_{\substack{y \in S \subseteq G \cup \{y\} \\ |S| = k}} \frac{1}{m+1} \binom{m}{k}^{-1} P[v, S] \right) \\
&= \left(\sum_{\substack{S \subseteq G \\ |S| = k}} \frac{m-k}{m+1} \cdot \frac{1}{m} \binom{m-1}{k}^{-1} P[v, S] \right) + \left(\sum_{\substack{S \subseteq G \\ |S| = k-1}} \frac{k}{m+1} \cdot \Delta_{v,y} \cdot \frac{1}{m} \binom{m-1}{k-1}^{-1} P[v, S] \right) \\
&= \frac{m-k}{m+1} \cdot \phi(v, G, k) + \frac{k}{m+1} \cdot \Delta_{v,y} \cdot \phi(v, G, k-1).
\end{aligned}$$

□

Lemma 8. *Let $v \in \mathcal{T}$ be a non-root node and let $Q \subseteq U \setminus \{d_{p_v}\}$. Then, for all k,*

$$\phi(v, Q, k) = \phi(p_v, Q, k) \cdot \frac{r_v}{r_{p_v}}.$$

Proof. The claim follows easily by the definition of $\phi(v, Q, k)$ and since $P[v, G] = P[p_v, G] \cdot \frac{r_v}{r_{p_v}}$ holds for every subset $G \subseteq Q$. □

Let $\Phi(v, G)$ be the sum of individual coordinates of the vector $\phi(v, G)$, i.e., $\Phi(v, G) := \sum_{k=0}^{|G|} \phi(v, G, k)$.

The following lemma states an intuitive fact that $\Phi(v, G)$ does not depend on the features in G that do not appear in the ancestors of v.

Lemma 9. *Let $v \in \mathcal{T}$ and $G \subseteq U$. Suppose $y \in U \setminus G \setminus F_v$. Then:*

$$\Phi(v, G \cup \{y\}) = \Phi(v, G).$$

Proof. Recall that $y \in U \setminus G \setminus F_v$ and thus $\Delta_{v,y} = 1$. By Lemma 7, we have:

$$\begin{aligned}
\Phi(v, G \cup \{y\}) &= \sum_{k=0}^{|G|+1} \phi(v, G \cup \{y\}, k) \\
&= \sum_{k=0}^{|G|+1} \frac{|G|+1-k}{|G|+2} \cdot \phi(v, G, k) + \sum_{k=0}^{|G|+1} \frac{k}{|G|+2} \cdot \phi(v, G, k-1) \\
&= \sum_{k=0}^{|G|} \frac{|G|+1-k}{|G|+2} \cdot \phi(v, G, k) + \sum_{k=0}^{|G|} \frac{k+1}{|G|+2} \cdot \phi(v, G, k) \\
&= \sum_{k=0}^{|G|} \phi(v, G, k) \\
&= \Phi(v, G).
\end{aligned}$$

□

The following is an analogue of Lemma 1 for Shapley value that reduces computing the Shapley explanation $(\phi_i)_{i \in U}$ to computing the vectors of the form $\phi(l, F_l \setminus \{i\})$ for all pairs $(l, i) \in \mathcal{L}(\mathcal{T}) \times U$ with $i \in F_l$.

Lemma 10. *For any $i \in U$, we have:*

$$\phi_i = \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} f(l) \cdot (\Delta_{l,i} - 1) \cdot \Phi(l, F_l \setminus \{i\}).$$

Proof. By expanding the sum (1) using (4), we obtain:

$$\begin{aligned}
\phi_i &= \frac{1}{n} \sum_{S \subseteq U \setminus \{i\}} \binom{n-1}{|S|}^{-1} (g(S \cup \{i\}) - g(S)) \\
&= \frac{1}{n} \sum_{S \subseteq U \setminus \{i\}} \binom{n-1}{|S|}^{-1} \left(\sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \left(P[l, S \cup \{i\}] - P[l, S] \right) \right)
\end{aligned}$$

By subsequently applying Lemma 2, and changing the summation order, we have:

$$\begin{aligned}
\phi_i &= \frac{1}{n} \sum_{S \subseteq U \setminus \{i\}} \binom{n-1}{|S|}^{-1} \left(\sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot P[l, S] \left(\Delta_{l,i} - 1 \right) \right) \\
&= \sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot (\Delta_{l,i} - 1) \left(\frac{1}{n} \sum_{k=0}^{n-1} \sum_{\substack{S \subseteq U \setminus \{i\} \\ |S| = k}} \binom{n-1}{k}^{-1} P[l, S] \right) \\
&= \sum_{l \in \mathcal{L}(\mathcal{T})} f(l) \cdot (\Delta_{l,i} - 1) \cdot \Phi(l, U \setminus \{i\})
\end{aligned}$$

Since $(\Delta_{l,i} - 1) = 0$ when $i \notin F_l$, we actually have:

$$\phi_i = \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} f(l) \cdot (\Delta_{l,i} - 1) \cdot \Phi(l, F_l \setminus \{i\}).$$

□

The recursive formulas of Lemmas 7 and 8 allow computing each $\phi(v, F_v)$ out of a "neighboring" vector $\phi(p_v, F_{p_v})$ in $O(|F_v|) = O(D)$ time. This overhead arises from the fact that the used vectors $\phi(\cdot, \cdot)$ have up to D coordinates. Recall that

when computing Banzhaf value explanations, similar values had only a single coordinate and hence a similar transition could be performed in constant time. Consequently, the basic algorithm of Section 3.1 adjusted to compute the vectors $\phi(v, F_v)$ takes $O(LD^2)$ time, which matches the bound achieved by Lundberg et al. [2020].

To obtain an asymptotically faster $O(LD)$ time algorithm for computing Shapley explanations using the approach of Section 3.2, we need to devise a Shapley-analogue of Lemma 5. To this end, consider the following values:

$$\Psi(v, k) = \sum_{l \in \mathcal{L}_v} f(l) \cdot \phi(l, F_l, k),$$

that are analogues of the values $B(v)$ from Section 3.2. By proceeding similarly as in Section 3.2, a bottom-up computation can be used to compute all the values $\Psi(v, k)$ for $v \in \mathcal{T}$ in $O(LD)$ time.

Let us also set:

$$\gamma(v, k) := \sum_{l \in \mathcal{L}_v} f(l) \cdot \phi(l, F_l \setminus \{d_{p_v}\}, k)$$
$$\Gamma(v) := \sum_{l \in \mathcal{L}_v} f(l) \cdot \Phi(l, F_l \setminus \{d_{p_v}\}).$$

Therefore, we can rewrite Lemma 10 as follows:

$$\begin{aligned}
\phi_i &= \sum_{\substack{l \in \mathcal{L}(\mathcal{T}) \\ i \in F_l}} f(l) \cdot (\Delta_{l,i} - 1) \cdot \Phi(l, F_l \setminus \{i\}) \\
&= \sum_{\substack{v \in \mathcal{T} \\ d_{p_v} = i}} \sum_{l \in \mathcal{L}_v} f(l) \cdot (\Delta_{l,i} - 1) \cdot \Phi(l, F_l \setminus \{i\}) \\
&= \sum_{\substack{v \in \mathcal{T} \\ d_{p_v} = i}} (\Delta_{v,i} - 1) \sum_{l \in \mathcal{L}_v} f(l) \cdot \Phi(l, F_l \setminus \{i\}) \\
&= \sum_{\substack{v \in \mathcal{T} \\ d_{p_v} = i}} (\Delta_{v,i} - 1) \cdot \Gamma(v).
\end{aligned}$$

Note that the above derivation provides an $O(L)$-time reduction of computing all ϕ_i to computing all values $\Gamma(v)$. Those can be clearly obtained by simple summation in $O(LD)$ time once we have all the values $\gamma(v, k)$.

The following lemma, analogous to Lemma 7, gives a relationship between values $\Psi(\cdot, \cdot)$ and $\gamma(\cdot, \cdot)$.

Lemma 11. *Let $v \in \mathcal{T}$, $v \neq \rho$. Suppose the sets F_l have equal sizes s for all $l \in \mathcal{L}_v$. Then, for any $k = 0, \ldots, s$, we have:*

$$\Psi(v, k) = \frac{s-k}{s+1} \cdot \gamma(v, k) + \frac{k}{s+1} \cdot \Delta_{v, d_{p_v}} \cdot \gamma(v, k-1).$$

Proof. By Lemma 7, for any $l \in \mathcal{L}_v$ we have:

$$\phi(l, F_l, k) = \frac{|F_l| - k}{|F_l| + 1} \phi(l, F_l \setminus \{d_{p_v}\}, k) - \frac{k}{|F_l| + 1} \cdot \Delta_{l, d_{p_v}} \cdot \phi(l, F_l \setminus \{d_{p_v}\}, k-1)$$
$$\phi(l, F_l, k) = \frac{s-k}{s+1} \phi(l, F_l \setminus \{d_{p_v}\}, k) - \frac{k}{s+1} \cdot \Delta_{l, d_{p_v}} \cdot \phi(l, F_l \setminus \{d_{p_v}\}, k-1).$$

We obtain the desired equality by summing the above through all $l \in \mathcal{L}_v$ and using $\Delta_{l, d_{p_v}} = \Delta_{v, d_{p_v}}$. □

Lemma 11 would suffice to compute all the needed values $\gamma(v, k)$ if only all the sets F_l, $l \in \mathcal{L}_v$ had equal sizes for each vertex $v \in \mathcal{T}$. Unfortunately, this is not true in general. To deal with this problem, we need to make a subtle change to the algorithm. Ideally, we would like all the sets F_l for $l \in \mathcal{L}(\mathcal{T})$ have the same size D, where D is the maximum size of F_l in the input tree. This could be ensured, for example, by extending all smaller F_l with $D - |F_l|$ distinct dummy features that do not appear in F_l – recall from Lemma 9 that adding dummy features does not change $\phi(v, G)$, for any $G \subseteq U$, so it

does not influence our results. Unfortunately, adding a dummy feature to F_l by simply using Lemma 7 costs $\Theta(D)$ time. Therefore, if T was very unbalanced, padding all F_l could cost as much as $\Theta(LD^2)$ time. We thus need a smarter approach.

Instead, let $q_1, \ldots, q_D$ be distinct artificial features *not* appearing in the nodes of $\mathcal{T}$. For *all* $v \in \mathcal{T}$ let us define

$$F_v^* = F_v \cup \{q_1, \ldots, q_{D-|F_v|}\}.$$

Observe that then $F_\rho^* = \{q_1, \ldots, q_D\}$ for the root ρ of $\mathcal{T}$, and for each non-root v we have

$$F_v^* = \begin{cases} F_{p_v}^* & \text{if } d_{p_v} \in F_{p_v} \\ F_{p_v}^* \setminus \{q_{D-|F_{p_v}|}\} \cup \{d_{p_v}\} & \text{otherwise.} \end{cases} \tag{6}$$

With sets F_v^* defined like this, $v \in \mathcal{T}$, by Lemma 9, we have:

$$\Phi(v, F_v \setminus \{d_{p_v}\}) = \Phi(v, F_v^* \setminus \{d_{p_v}\}),$$

and consequently:

$$\Gamma(v) = \sum_{l \in \mathcal{L}_v} f(l) \cdot \phi(l, F_l^* \setminus \{d_{p_v}\}).$$

It is thus enough to modify the basic algorithm computing all the vectors $\psi(v, F_v)$ so that it computes all the vectors $\phi(v, F_v^*)$ instead. It is very easy to achieve that. First of all, the initial vector $\phi(\rho, F_\rho^*)$ is initialized in $O(D^2) = O(LD)$ time by applying Lemma 7 D times. By (6), for each non-root v, the vector $\phi(v, F_v^*)$ can be still obtained from the vector $\phi(p_v, F_{p_v}^*)$ in $O(D)$ time as before using $O(1)$ applications of Lemmas 7 and 8.

F PROOF OF LEMMA 6

Let us first argue that indeed moving between nearby values $\beta(v, G)$ boils down to $O(1)$ multiplications/divisions of some state value $\beta(v, G)$ with a number between 0.5 and r_ρ. Indeed, if Lemma 3 is used, then $\beta(v, G)$ is multiplied by a number that is at least 0.5 (if $x_f \notin I_{v,f}$), and at most $(1 + 1/c_v(f))/2 \leq (1 + 1/(1/r_\rho))/2 \leq (1 + r_\rho)/2 \leq r_\rho$, since the coverages r_v are positive integers. On the other hand, Lemma 4 requires a single multiplication via a number of the form r_v/r_{p_v}, which translates to two multiplications/divisions by an integer between 1 an r_ρ.

Let $\epsilon < 0.1$ be the machine epsilon. Using a well-established model of floating point numbers (see e.g., Higham [2002]) , we can assume that the floating point number representation $\text{fl}(x \circ y)$ of the result of an arithmetic operation $\circ$ on two *exactly represented* numbers x, y satisfies $\text{fl}(x \circ y) = (x \circ y)(1 + \delta)$ for some $|\delta| \leq \epsilon$. In particular, if $x, y > 0$ and $\circ \in \{+, \cdot, /\}$, then we have

$$\begin{aligned} &\text{fl}(x \circ y) \leq (x \circ y)(1 + \epsilon), \\ &\text{fl}(x \circ y) \geq (x \circ y)(1 - \epsilon) \geq (x \circ y) \cdot \frac{1}{(1 + \epsilon)^2}. \end{aligned}$$

We can thus conclude, that if $x' > 0$ is floating-point approximation of a value $x > 0$ with multiplicative error between $(1 + \epsilon)^{-k}$ and $(1 + \epsilon)^k$, and $y' > 0$ is a floating-point approximation of a value $y > 0$ with multiplicative error between $(1 + \epsilon)^{-l}$ and $(1 + \epsilon)^l$, then for any $\circ \in \{+, \cdot, /\}$ we have:

$$\begin{aligned} &\text{fl}(x' \circ y') \leq (x' \circ y')(1 + \epsilon) \leq (x \circ y)(1 + \epsilon)^{\max(k,l)} \cdot (1 + \epsilon) = (x \circ y)(1 + \epsilon)^{\max(k,l)+1}, \\ &\text{fl}(x' \circ y') \geq (x' \circ y')(1 - \epsilon) \geq (x \circ y)\frac{1}{(1 + \epsilon)^{\max(k,l)}} \cdot \frac{1}{(1 + \epsilon)^2} = (x \circ y)\frac{1}{(1 + \epsilon)^{\max(k,l)+2}}. \end{aligned}$$

More generally, evaluating an arithmetic expression on positive numbers that involves only additions, multiplications, or divisions, built of k such operations, has multiplicative error at most $(1 + \epsilon)^{O(k)}$ and at least $(1 + \epsilon)^{-O(k)}$. Note that this implies a relative error of $(1 + \epsilon)^{O(k)} - 1$. Indeed, if the expression evaluates to $z' > 0$ and its true value is $z > 0$, then if $z' \geq z$, we get

$$\frac{|z' - z|}{z} = \frac{z' - z}{z} \leq \frac{(1 + \epsilon)^{O(k)} z - z}{z} = (1 + \epsilon)^{O(k)} - 1,$$

whereas if $z' \leq z$, then we get (by applying, in the final step, the inequality $x + 1/x \geq 2$ valid for all $x > 0$):

$$\frac{|z' - z|}{z} = \frac{z - z'}{z} \leq \frac{z - (1+\epsilon)^{-O(k)} z}{z} = 1 - \frac{1}{(1+\epsilon)^{O(k)}} \leq (1+\epsilon)^{O(k)} - 1.$$

Finally, note that each value $\beta(v, G)$ for v at depth d, can be expressed, by an inductive application of Lemmas 3 and 4, using a formula with $O(d + |G|)$ multiplications and divisions and input values in the range $[0.5, r_\rho]$, all of which can be represented as floating point numbers with multplicative error between $(1+\epsilon)^{-1}$ and $(1+\epsilon)$. As a result, since $d \leq D$, and $|F_l| \leq D$, for any leaf l, and any $i \in F_l$, $\beta(l, F_l)$ is computed using $O(D)$ applications of Lemmas 3 and 4, and thus with relative error $(1+\epsilon)^{O(D)} - 1$.