# Accurate Estimation of Feature Importance Faithfulness for Tree Models

**Mateusz Gajewski** [*2, 3], **Adam Karczmarz** [*1, 2], **Mateusz Rapicki** [*1], **Piotr Sankowski**[1, 4]

[1] Faculty of Mathematics, Informatics and Mechanics University of Warsaw, Warsaw, Poland
[2]IDEAS NCBR
[3] Faculty of Computing and Telecommunications Poznan University of Technology, Poznan, Poland
[4] MIM Solutions
mg96272@gmail.com, a.karczmarz@mimuw.edu.pl, mateusz.rapicki@mimuw.edu.pl, sank@mimuw.edu.pl

## Abstract

In this paper, we consider a perturbation-based metric of predictive faithfulness of feature rankings (or attributions) that we call *PGI squared*. When applied to decision tree-based regression models, the metric can be computed *exactly* and efficiently for arbitrary independent feature perturbation distributions. In particular, the computation does not involve Monte Carlo sampling that has been typically used for computing similar metrics and which is inherently prone to inaccuracies.

As a second contribution, we proposed a procedure for constructing feature ranking based on PGI squared. Our results indicate the proposed ranking method is comparable to the widely recognized SHAP explainer, offering a viable alternative for assessing feature importance in tree-based models.

**Code** — https://github.com/rapicki/prediction-gap
**Extended version** — https://arxiv.org/abs/2404.03426

## Introduction

One of the key challenges in deploying modern machine learning models in such areas as medical diagnosis lies in the ability to indicate why a certain prediction has been made. Such an indication may be of critical importance when a human decides whether the prediction can be relied on. This is one of the reasons various aspects of explainability of machine learning models have been the subject of extensive research lately (see, e.g., (Burkart and Huber 2021)).

For some basic types of models (e.g., single decision trees, the rationale behind a prediction is easy to understand by a human. However, predictions of more complex models (that offer much better accuracy, e.g., based on neural networks or decision tree ensembles) are also much more difficult to interpret. Accurate and concise explanations understandable to humans might not always exist. In such cases, it is still beneficial to have methods *giving a flavor* of what factors might have influenced the prediction the most.

Local feature attribution[1] constitutes one of such general approaches. For a fixed input $x$ and the model's prediction $f(x)$, each feature is assigned a weight "measuring" the feature's impact on the prediction. Ideally, a larger (absolute) weight should correspond to a larger importance of the feature for the prediction.

Many local attribution methods have been proposed so far, e.g., (Lundberg and Lee 2017; Plumb, Molitor, and Talwalkar 2018; Ribeiro, Singh, and Guestrin 2016). They are often completely model-agnostic, that is, they can be defined for and used with any model which is only accessed in a black-box way for computing the attribution. In some of these cases, focusing on a particular model's architecture can lead to much more efficient and accurate attribution computation algorithms (e.g., (Lundberg et al. 2020) for tree ensemble models).

With the large body of different attribution methods, it is not clear which one should one use, especially since there is not a single objective measure of their reliability. To help deal with this issue, several notions of explanation quality have been proposed in the literature, such as faithfulness (fidelity) (Liu et al. 2021; Yeh et al. 2019), stability (Alvarez-Melis and Jaakkola 2018), or fairness (Balagopalan et al. 2022; Dai et al. 2022). Agarwal et al. (2022) developed an open-source benchmark OpenXAI automatically computing variants of these measures for a number of proposed attribution methods.

Out of these notions, our focus in this paper is on faithfulness. Roughly speaking, an attribution method deserves to be called faithful if the features deemed important by the attribution are truly important for the decision-making process of the model. The *perturbation-based* methods constitute one popular class of approaches to measuring faithfulness. Intuitively, perturbing features deemed impactful should generally lead to a significant change in the prediction. On the contrary, manipulating unimportant features should not make a big difference.

One concrete cleanly-defined perturbation-based faithfulness metric for regression problems is the *prediction gap on important feature perturbation*, PGI in short. PGI is a faithfulness measure of choice in the OpenXAI benchmark (Agarwal et al. 2022). Having fixed some subset of important features $S$ derived from the obtained attribution, the *prediction gap* (PG) of $x$ wrt. $S$ is defined as:

$$\mathrm{PG}(x, S) := \mathbb{E}_{x' \sim \mathrm{perturb}(x,S)} \left[ |f(x') - f(x)| \right]. \quad (1)$$

In (Agarwal et al. 2022), the concrete perturbation method

[*]These authors contributed equally.

[1]It should be contrasted with *global* feature attribution whose aim is to measure the feature's total impact on the model.

perturb$(x, S)$ is adding an independent Gaussian noise from $\mathcal{N}(0, \sigma^2)$ to each coordinate $x_j$, $j \in S$. $\mathrm{PGI}(x)$ can be defined as either the prediction gap wrt. a fixed set of important features (e.g., $k$ top-scoring features for some fixed $k$ (Dai et al. 2022)), or an average prediction gap over many important feature sets (as in (Agarwal et al. 2022)). In particular, the PGI metric depends *only* on the ordering of features by importance induced by the attribution. Other perturbation-based methods include e.g., PGU (Agarwal et al. 2022), comprehensiveness, sufficiency (DeYoung et al. 2020), remove-and-retrain (Hooker et al. 2019), deletion (Petsiuk, Das, and Saenko 2018).

In practice, PGI and other methods based on random perturbations are computed using the Monte Carlo method which is inherently prone to inaccuracies, unless a large number of samples is used.

It is interesting to ask whether PGI or similar random perturbation-based faithfulness metrics can be computed *exactly*, that is, via a closed-form calculation, so that any approximation error is caused merely by the floating-point arithmetic's rounding errors.

## Our Contribution

**Exact Computation** First, we consider random perturbation-based measurement of faithfulness from the point of view of efficient and accurate computation. Of course, it is unrealistic to assume that such a quantity can be computed exactly (or even beyond the Monte Carlo method) given only black-box access to the model and the perturbation distribution. Hence, we focus on a concrete model architecture: *tree ensemble models*. Tree ensemble models remain a popular choice among practitioners since they are robust, easy to tune, and fast to train.

Unfortunately, computing $\mathrm{PG}(x, S)$ (wrt. a fixed set $S$ of perturbed features, as in (Agarwal et al. 2022), see (1)) does not look very tractable even given the rich structure that tree ensemble models offer. This is because therein, the expectation is taken over the *absolute value* of $f(x) - f(x')$, where $x'$ is $x$ with important features perturbed. The absolute value is not very mathematically convenient to work with when taking expectations. On the other hand, dropping the absolute value here would not make much sense: a small random perturbation (positive or negative) of a feature could in principle lead to a large positive or negative prediction difference. These, in turn, could cancel out in expectation, making the metric close to zero even though the feature is visibly of high importance.

*Theoretical contribution:* To deal with the tractability problem of measuring PGI on tree models, we propose a similar metric, $\mathrm{PGI}^2$, obtained from a slightly different *squared prediction gap*:

$$\mathrm{PG}^2(x, S) := \mathbb{E}_{x' \sim \mathrm{perturb}(x,S)} \left[ (f(x') - f(x))^2 \right]. \quad (2)$$

This eliminates the cancellation problem outlined above. Of course, introducing the square can alter the evaluation of the feature's importance wrt. PGI as defined previously. However, intuitively, it amplifies the big prediction differences, and at the same time marginalizes the small, which is a generally good property if we seek concise explanations.

We prove that for tree ensemble models with $n$ nodes in total, the squared prediction gap $\mathrm{PG}^2(x, S)$ can be computed in $O(n^2)$ time *exactly* for any fixed choice of a subset $S$ of perturbed important features, assuming the cumulative distribution function of feature-wise random perturbations can be evaluated in constant time. That is, the obtained running time bound does not require that the random perturbations are Gaussian or their distributions are equal across features.

We note that whereas our $\mathrm{PG}^2$ algorithm is quadratic, from the theoretical point of view, the important thing is that on tree models, a prediction gap-style quantity computation can be carried out *exactly* in *polynomial time* after all.

Given a permutation $\pi$ ranking the $d$ features from the most to least important, the OpenXAI benchmark calculates PGI by taking the average prediction gap obtained when perturbing $k = 1, 2, \ldots, d$ most important features $\pi[1..k] = \{\pi(1), \ldots, \pi(k)\}$ (see (Agarwal et al. 2022, Appendix A)). In such a case, $d$ different expectations of absolute prediction gap are estimated, each via Monte Carlo sampling. In our case, $\mathrm{PGI}^2(x)$ is analogously defined as:

$$\mathrm{PGI}^2(x, \pi) := \frac{1}{d} \sum_{k=1}^{d} \mathrm{PG}^2(x, \pi[1..k]). \quad (3)$$

Using our prediction gap algorithm, $\mathrm{PGI}^2$ is computed exactly (assuming infinite precision arithmetic; in reality, up to the precision error incurred by the usage of floating-point arithmetic) in $O(n^2 d)$ time.

*Experiments:* In our experiments, we evaluated different methods for calculating the Prediction Gap. Our study encompassed the exact algorithm as well as two sampling techniques integration techniques: Monte Carlo (MC) and Quasi-Monte Carlo (QMC). With the increase in iteration count, the outputs of MC and QMC visibly converged to the output of our exact algorithm; this confirms good numerical stability of our approach.

When allocating an equivalent time budget across all methods, our findings revealed an advantage over MC and QMC, with QMC demonstrating a slight edge. Specifically, the Normalized Mean Absolute Error (NMAE) for MC was 0.13 for single models and decreased to 0.01 for bigger models. In comparison, QMC exhibited NMAEs of approximately 0.05 for single models and 0.002 for bigger ones. These results underscore the efficacy of sampling methods in the context of computing the PG, particularly for more sophisticated model structures.

**$\mathrm{PG}^2$-Based Greedy Feature Ordering.** We also investigate the possibility of using our exact $\mathrm{PG}^2$ algorithm as a base of a feature importance ranking algorithm. We stress that finding a feature ordering optimizing the $\mathrm{PGI}^2$ metric is a highly non-trivial task, as the metric depends on the entire ordering of the features. Checking all the features' permutations is infeasible in most cases.

We consider constructing the feature ranking using a *greedy* $\mathrm{PG}^2$ heuristic: the $i$-th most important feature $\lambda$ is chosen so that it optimizes the squared prediction gap together with the already chosen $i - 1$ most important features

plus $\lambda$. We next compare the greedy $\mathrm{PG}^2$ ranking with the ranking produced using the popular SHAP feature attribution method for tree ensembles (Lundberg et al. 2020). We observe that the most important features identified by the two methods generally differ, and the deviation is clearer if the applied perturbations are smaller.

As far as faithfulness is considered, our experiments confirm that the greedy $\mathrm{PG}^2$ ranking yields, on average, better $\mathrm{PGI}^2$ scores than SHAP. While that such a property holds is non-obvious, it is perhaps not very surprising either. This is why we also compare the $\mathrm{PG}^2$-based ranking method with SHAP in terms of two different faithfulness metrics: *feature randomization* and *remove-and-retrain* (Hooker et al. 2019).

In the former case and some of our datasets, the $\mathrm{PG}^2$-based rankings (used with Gaussian perturbations for small standard deviations) performed better than SHAP, whereas in the others, the results were comparable. Wrt. the latter metric, SHAP achieved better performance.

## Preliminaries

Let us denote by $f : \mathbb{R}^d \to \mathbb{R}$ the output function of the considered regression model. We sometimes also use $f$ to refer to the model itself. The input $x \in \mathbb{R}^d$ to $f$ is called a feature vector, and we denote by $x_i$ the value of the $i$-th feature of $x$. Thus, we identify the set of features with $[d]$.

**Tree Models.** When talking about decision trees, we assume them to be binary and based on single-value splits. That is, each non-leaf node $v$ of a decision tree $\mathcal{T}$ has precisely two children $a_v, b_v$. It is also assigned a feature $q_v \in [d]$ and a threshold value $t_v \in \mathbb{R}$. Each leaf node $l \in \mathcal{T}$ is in turn assigned a value $y_v \in \mathbb{R}$. We denote by $\mathcal{L}(\mathcal{T})$ the set of leaves of the tree $\mathcal{T}$.

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 either to the child $a_v$ if $x_{q_v} < t_v$, or to $b_v$ otherwise. When a leaf $l$ is eventually reached, its value $y_l$ is returned. We denote by $n_{\mathcal{T}}$ the number of nodes in $\mathcal{T}$.

When considering tree ensemble models $(\mathcal{T})_{i=1}^m$, the output $f(x)$ of the model is simply the sum of outputs $f_{\mathcal{T}_i}(x)$ of its $m$ individual trees. We generally use $n = \sum_{i=1}^m n_{\mathcal{T}_i}$ to refer to the *total size* of the tree ensemble model.

## Computing $\mathrm{PG}^2$ for Tree Ensemble Models

Let $S \subseteq [d]$ be a subset of *important features* and $x \in \mathbb{R}^d$. In this section, we show our polynomial exact algorithm computing $\mathrm{PG}^2(x, S)$, as defined in (2).

We consider distributions $\mathrm{perturb}(x, S)$ such that $x' \sim \mathrm{perturb}(x, S)$ satisfies:

1. for every *perturbed feature* $i \in S$, $x'_i = x_i + \delta_i$, where $\delta_i \sim \mathcal{D}_i$, and distribution $\mathcal{D}_i$ is fixed, and the c.d.f. $F_{\mathcal{D}_i}$ of $\mathcal{D}_i$ can be evaluated in $O(1)$ time.
2. for every *non-perturbed feature* $i \notin S$, $x'_i = x_i$,
3. all $\delta_j$ ($j \in S$) are independent random variables.

The main goal of this section is to show how to compute

$$\mathrm{PG}^2(x, S) := \mathbb{E}_{x' \sim \mathrm{perturb}(x,S)} \left[ (f(x') - f(x))^2 \right]$$

under the assumptions made in the case when $f$ is a tree ensemble model $(\mathcal{T})_{i=1}^m$. Recall from Equation (3) that the metric $\mathrm{PGI}^2(x, \pi)$ for a given ranking $\pi$ can be computed by running the algorithm $d$ times for $d$ different subsets $S$ of the form $\pi[1..k]$.

Put $c := f(x)$. Let $x' \sim \mathrm{perturb}(x, S)$. Recall that $f(x') = \sum_{i=1}^m f_{\mathcal{T}_i}(x')$. For every leaf node $v$ of some $\mathcal{T}_i$, consider a random indicator variable $X_v$ such that $X_v = 1$ iff $\mathcal{T}_i$ evaluates to the value $y_v$ of leaf $v$ given $x'$ as input. In particular, for any $i = 1, \ldots, m$, there is exactly one $v \in \mathcal{L}(\mathcal{T}_i)$ such that $X_v = 1$. Set $\mathcal{L} = \bigcup_{i=1}^m \mathcal{L}(\mathcal{T}_i)$. Then:

$$f(x') = \sum_{v \in \mathcal{L}} X_v \cdot y_v.$$

By linearity of expectation, we obtain:

$$\begin{aligned}
\mathrm{PG}^2(x, S) &= \mathbb{E}\left[ \left( \sum_{v \in \mathcal{L}} X_v y_v - c \right)^2 \right] \\
&= c^2 + \mathbb{E}\left[ \left( \sum_{u \in \mathcal{L}} X_u y_u \right) \left( \sum_{v \in \mathcal{L}} X_v y_v - 2c \right) \right] \\
&= c^2 + \sum_{\substack{u,v \in \mathcal{L} \\ u \neq v}} y_u y_v \Pr[X_u = 1 \wedge X_v = 1] \\
&\qquad + \sum_{u \in \mathcal{L}} \Pr[X_u = 1] \cdot y_u \cdot (y_u - 2c).
\end{aligned}$$

The above formula reduces computing $\mathrm{PG}^2(x, S)$ to calculating probabilities of the form $\Pr[X_u = 1 \wedge X_v = 1]$ (where $u \neq v$) or of the form $\Pr[X_u = 1]$. We focus on the former task as dealing with the latter is analogous but easier.

For a node $w$, let $Q_w$ be the set of features appearing on the root-$w$ path (in the tree whose $w$ is the node of). Observe that for $w \in \mathcal{L}$, $X_w = 1$ holds iff for each feature $q \in Q_w$, $x'_q$ falls into a certain interval $I_{w,q}$. Namely, if the root-to-$w$ path consists of nodes $w_1, \ldots, w_k = w$, then $I_{w,q}$ can be constructed as follows. Start with the interval $(-\infty, \infty)$, and follow the path downwards. For each encountered node $w_i$ splitting on $q$, intersect the current $I_{w,q}$ with either:

- $(-\infty, t_{w_i})$ if $w_{i+1} = a_{w_i}$, or
- $[t_{w_i}, \infty)$ if $w_{i+1} = b_{w_i}$, or
- $\emptyset$ otherwise.

It follows that the event $X_u = 1 \wedge X_v = 1$ occurs iff for all $q \in Q_u \cup Q_v$, $x'_q \in I_{u,q} \cap I_{v,q}$. Hence, if we denote by $l_{u,v,q}, r_{u,v,q}$ the respective endpoints of $I_{u,q} \cap I_{v,q}$, by the independence of all perturbations $\delta_j$, we obtain that $\Pr[X_u = 1 \wedge X_v = 1]$ equals $\Pi(u, v)$, where

$$\Pi(u, v) = \left( \prod_{q \in (Q_u \cup Q_v) \setminus S} [x_q \in I_{u,q} \cap I_{v,q}] \right) \cdot \prod_{q \in (Q_u \cup Q_v) \cap S} \left( F_{\mathcal{D}_q}(r_{u,v,q} - x_q) - F_{\mathcal{D}_q}(l_{u,v,q} - x_q) \right). \tag{4}$$

Implemented naively, computing a single value $\Pi(u, v)$ takes $O(D_u + D_v)$ time, where $D_u, D_v$ are the depths of

$u$ and $v$ in their respective trees. Indeed, note that by traversing the root-to-$u$ and root-to-$v$ paths, we can find all the relevant intervals $I_{u,q}, I_{v,q}$ that the product $\Pi(u, v)$ depends on. Moreover, $\Pi(u, v)$ has at most $D_u + D_v$ factors, each of which can be evaluated in $O(1)$ time.

Computing the values $\Pi(u, v)$ through all $\Theta(n^2)$ pairs $u, v \in \mathcal{L}$ could take $O(n^2 D)$ time, where $D$ is a global bound on the depth of the trees in the ensemble.

However, the computation can be optimized to run in $O(n^2)$ time by computing the probabilities in an adequate order. Namely, note that, say, the value $\Pi(a_u, v)$ can be computed based on $\Pi(u, v)$ in $O(1)$ time. This is because the interval $I_{a_u,q}$ can differ from the corresponding interval $I_{u,q}$ only for the feature $q = q_u$, and it can be obtained by splitting $I_{u,q}$ with the threshold $t_u$. Hence, only one factor of the product (4) has to be added or replaced in order to obtain $\Pi(a_u, v)$ from $\Pi(u, v)$. An analogous trick applies to computing $\Pi(b_u, v)$, $\Pi(u, a_v)$, $\Pi(u, b_v)$ from $\Pi(u, v)$. Consequently, by iterating through nodes $u$ in pre-order, and then, once $u \in \mathcal{L}$ is fixed, through nodes $v$ in pre-order (the order of processing trees does not matter), all the required values $\Pi(u, v)$ can be computed in $O(n^2)$ total time.

This idea is the realized in Algorithm 1, where, while the two nested pre-order traversals over $\mathcal{T}$ proceed, the endpoints of intervals $I_{\cdot,\cdot}$ are maintained using the arrays $l, r$, and the factors of (4) are maintained using the array factor.

**Theorem 0.1.** *Let $f$ be a tree ensemble model whose trees have $n$ nodes in total. Let $x \in \mathbb{R}^d$. Then for any $S \subseteq [d]$, $\mathrm{PG}^2(x, S)$ can be computed in $O(n^2)$ time.*

## Experiments: Exact vs Monte Carlo Sampling

In this section, we compare our algorithm computing the squared prediction gap (as defined in Equation (2)) for tree ensembles as described earlier, the regular Monte Carlo sampling-based method (MC) and Quasi Monte Carlo sampling method (QMC)[2]. Specifically, the Monte Carlo methods, given $x$ and the set $S \subseteq [d]$ or perturbed features, evaluates $f(x')$ for $i$ randomly sampled $x' \sim \mathrm{perturb}(x, S)$ and records the average value of $(f(x') - f(x))^2$ over the samples. We will call $i$ the number of *iterations* in the following.

Recall that the accuracy of Monte Carlo methods is supposed to increase with the number of iterations $i$. The subject of the comparison was hence the accuracy of the results produced by the sampling algorithms as a function of the number of iterations with the goal of deciding which method (exact, MC, or QMC) is preferable and under what conditions.

Specifically, having fixed a model $f$, we looked at the *average difference* between the values $\mathrm{PG}^2(x, S)$ computed using our closed-form algorithm and the two considered sampling methods. Using the output of our algorithm as the "ground truth" requires the algorithm to be numerically stable. So an indirect goal of the comparison was to establish whether our algorithm has good numerical properties.

[2]The details of quasi-random generators used in our Quasi Monte Carlo implementation can be found in the Appendix.

Algorithm 1: Computing $\Pi(u, v)$ for all leaf pairs $u, v$ given model $\mathcal{T}$, a feature vector $x$, and important features $S \subseteq [d]$

**Require:** $\forall_{q \in [d]} \quad l[q] = \infty, r[q] = \infty, \mathrm{factor}[q] = 1$

**procedure** COMPUTE-PROB($v, \mathrm{prod} = 1, u = \mathbf{null}$)
  **if** $v$ is non-root **then** ▷ update the maintained arrays
    $p :=$ parent of $v$ in $\mathcal{T}$
    $q := q_p$
    $(l', r', f') := (l[q], r[q], \mathrm{factor}[q])$
    **if** $v = a_p$ **then** ▷ $v$ is the left child of its parent
      $r[q] := \min(r[q], t_p)$
    **else**
      $l[q] := \max(l[q], t_p)$
    **end if**
    **if** $q \in S$ **then** ▷ See Equation (4)
      $\mathrm{factor}[q] := F_{\mathcal{D}_q}(r[q] - x_q) - F_{\mathcal{D}_q}(l[q] - x_q)$
    **else**
      $\mathrm{factor}[q] := [x_q \geq l[q] \wedge x_q \leq r[q]]$
    **end if**
    $\mathrm{prod} := (\mathrm{prod}/f') \cdot \mathrm{factor}[q]$
  **end if**
  **if** $v$ is a leaf **then**
    **if** $u =$ **null then**
      **for** $T \in \mathcal{T}$ **do** ▷ inner loop through trees
        COMPUTE-PROB(root$(T)$, prod, $v$)
      **end for**
    **else** ▷ both leaves $u$ and $v$ are fixed
      $\Pi(u, v) := \mathrm{prod}$
    **end if**
  **else** ▷ descend
    COMPUTE-PROB($a_v$, prod, $u$)
    COMPUTE-PROB($b_v$, prod, $u$)
  **end if**
  **if** $v$ is non-root **then** ▷ revert changes to $l, r, \mathrm{factor}$
    $(l[q], r[q], \mathrm{factor}[q]) := (l', r', f')$
  **end if**
**end procedure**

**for** $T \in \mathcal{T}$ **do** ▷ outer loop through trees
  COMPUTE-PROB(root$(T)$)
**end for**

### Experimental Setup

**Datasets** Following the previous work (e.g., (Agarwal et al. 2022; Dai et al. 2022)), in each experiment we assume that the (real) perturbations used for computing the prediction gap (either exactly, or approximately) come from the same normal distribution $\mathcal{D} = \mathcal{N}(0, \sigma^2)$. This is why we need to assume that all the features have numerical values.

Using the same perturbation distribution for all the features requires the features be standardized by subtracting the mean and dividing by the standard deviation. This ensures that if a feature is perturbed by a noise variable drawn from $\mathcal{D}$, the perturbation will have a similar effect. More specifically, it prevents a perturbation from $\mathcal{D}$ being relatively significant for one feature and insignificant for another.

In our experiments, we used the following datasets:

1. Red Wine Quality (Cortez et al. 2009). The dataset con-

tains 11 features wine, all numerical and continuous. The task is to predict the score of a wine, which is an integer between 1 and 10. We considered it as a regression task. The dataset contains 1 599 examples and has 11 features.

2. California Housing (Torgo 2023) The dataset contains information from the 1990 Californian census. There are 8 numerical characteristics, and one categorical - proximity to the ocean. For the reasons outlined before, we decided to drop this feature and use a modified dataset. The task is to predict the median value of the house. The dataset contains 20 640 examples and has 8 features.
3. Parkinson Telemonitoring Data (Tsanas and Little 2009) The dataset contains 5 875 voice measurements from Parkinson's disease patients, collected at home. It includes 17 numerical features, after dropping 3 categorical columns(ID, age) with a task to predict UPDRS motor and total scores.

**Models** All tree ensembles are implemented in the XGBoost library (Chen and Guestrin 2016). For each of the datasets, models of the following two types were trained:

- Single tree – one decision tree with `max_depth` up to 4.
- Bigger – gradient boosted trees with `max_depth` up to 4 and number of trees limited to 40.

In each case, the data set was split 80:20 into training and test sets. In addition, for each model and its constraint, a grid search was performed over selected hyperparameters. The details of the training process can be found in the Appendix.

**Measurements** The comparison was carried out in different settings. For a selected dataset and sampling method (MC/QMC), the variable parameters were:

- Model type $m$. Recall that, in our case, there were two model types for a fixed dataset.
- The standard deviation $\sigma$ of a Gaussian used to perturb a feature. We used the values $\{0.1, 0.3, 1.0\}$.

To carry out the experiment having fixed the above parameters, we used the following procedure. For each number of iterations $i \in \{100, 500, 1000, 2000, 4000, 6000, 8000, 10000, 15000, 20000, 25000, 30000, 35000\}$, we ran our closed-form algorithm and the sampling method in question with iteration count $i$, both estimating the same value $\mathrm{PG}^2(x, S)$ over $N = 20\ 000$ random pairs $(x, S)$. For each such pair, we recorded: (1) the average difference between the two estimates and (2) the computation times. The $N$ pairs varied in the size of the perturbed feature set $S$ (which could be, e.g., $1, \ldots, 11$ for the Red Wine Quality dataset), and we ensured that all subset sizes were equally represented among the pairs. Each pair involved a randomly selected point from a test dataset and a random set of features of the desired size.

**Implementation** All algorithms were implemented in Python with the key parts including inference on the tree ensemble being implemented as a C++ package and all the arithmetic operations in the algorithms were carried out using the `numpy.float32` type or C++ `float` type.

The computations were carried out on a FormatServer THOR E221 (Supermicro) server equipped with two AMD EPYC 7702 64-Core processors and 512 GB of RAM with operation system Ubuntu 22.04.1 LTS.

## Results

**Accuracy** To estimate the accuracy of sampling-based methods as a function of the iteration count, we computed the Normalised Mean Absolute Error (NMAE), defined as:

$$\mathrm{NMAE}(y, \hat{y}) = \frac{\sum_{j=1}^{N} \mid y_j - \hat{y}_j \mid}{\sum_{j=1}^{N} \mid y_j \mid},$$

where $y_j$ is the respective value $\mathrm{PG}(x_j, S_j)$ computed by the exact algorithm, and $\hat{y}_j$ is the respective estimate obtained by the sampling algorithm. As opposed to the more popular MAPE error, NMAE was used because it is well defined in cases when true prediction is 0, which is often the case (for example when the perturbation is small).

The results obtained in the experiment – the error as a function of the number of iterations – are depicted using the "Mean Monte Carlo" and "Mean Quasi Monte Carlo" curves for $\sigma = 0.3$ and Bigger model for the Red Wine Quality dataset in Figure 1. Analogous figures for other datasets, models and $(m, \sigma)$ can be found in the Appendix. Overall,

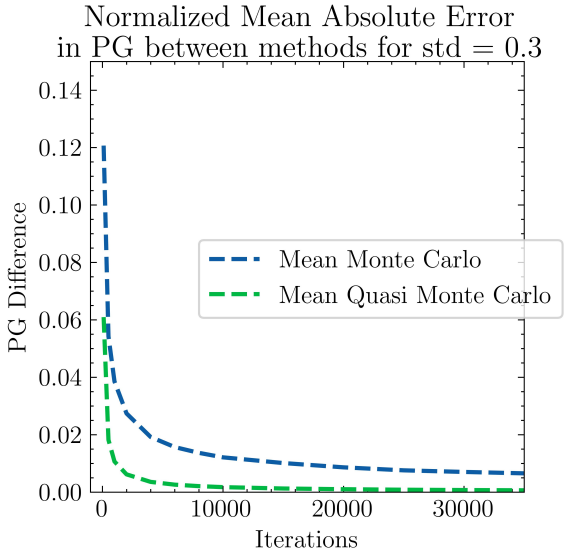


Figure 1: NMAE for $\sigma = 0.3$ and for Bigger model for Red Wine Quality dataset.

the difference between the exact and sampling algorithms decreases as the number of iterations in the sampling algorithm increases. This is indeed anticipated since MC sampling yields better accuracy with iteration increase; in fact, the relative error decreases proportionally to $1/\sqrt{i}$. Additionally, the error for the QMC method is significantly lower, especially for the lower count of iterations. Moreover, the consistent $\mathrm{NMAE}(y, \hat{y})$ error drop with the increase of the iteration count of the sampling methods indicates that our closed-form algorithm is numerically stable.

**Efficiency** The running time of the sampling methods depends, clearly, linearly on the iterations count. As a result, the decision about which method to choose should depend on the acceptable accuracy. To this end, for each of the considered settings (dataset and parameters), we identified the number of iterations (out of 13 considered thresholds as explained above) in the respective sampling algorithm that came closest to the running time of the exact algorithm for each setting. These corresponding numbers of iterations and the associated NMAEs are shown in Table 1. A larger acceptable NMAE than the corresponding threshold value suggests using the respective sampling-based method, and for smaller NMAEs than that, the exact algorithm is preferable.

| | **Monte Carlo** | | **Quasi Monte Carlo** | |
|---|---|---|---|---|
| $\sigma$ | Iterations | NMAE | Iterations | NMAE |
| Single tree Red Wine Quality model | | | | |
| 0.1 | 100 | 0.132 | 100 | 0.04 |
| 0.3 | 100 | 0.136 | 100 | 0.049 |
| 1.0 | 100 | 0.122 | 100 | 0.054 |
| Bigger Red Wine Quality model | | | | |
| 0.1 | 4000 | 0.019 | 4000 | 0.003 |
| 0.3 | 8000 | 0.014 | 8000 | 0.002 |
| 1.0 | 15000 | 0.009 | 15000 | 0.001 |
| Single California Housing model | | | | |
| 0.1 | 100 | 0.158 | 100 | 0.051 |
| 0.3 | 100 | 0.14 | 100 | 0.043 |
| 1.0 | 100 | 0.129 | 100 | 0.049 |
| Bigger California Housing model | | | | |
| 0.1 | 4000 | 0.018 | 4000 | 0.002 |
| 0.3 | 10000 | 0.011 | 10000 | 0.001 |
| 1.0 | 15000 | 0.01 | 15000 | 0.001 |
| Single Parkinson Telemonitoring model | | | | |
| 0.1 | 500 | 0.071 | 100 | 0.037 |
| 0.3 | 500 | 0.064 | 500 | 0.011 |
| 1.0 | 500 | 0.085 | 100 | 0.092 |
| Bigger Parkinson Telemonitoring model | | | | |
| 0.1 | 6000 | 0.015 | 4000 | 0.004 |
| 0.3 | 15000 | 0.009 | 15000 | 0.002 |
| 1.0 | 20000 | 0.008 | 20000 | 0.002 |

Table 1: Iterations needed to match the execution time of our exact algorithm, and the corresponding NMAEs.

We observe that the vanilla Monte Carlo simulation gives noticeably less accurate results than our exact method with a comparable allocated time budget. We observe that the error generally decreases when increasing the size of the model. This is justified by the fact that our algorithm's running time increases quadratically with $n$, whereas a single iteration of the Monte Carlo and Quasi Monte Carlo method runs in time at most linear in $n$. As a result, the bigger the model size is, the more iterations the Monte Carlo and Quasi Monte Carlo methods can perform within the same time budget. Moreover, for Monte Carlo the error decreases with the standard deviation of the distribution used to obtain perturbations.

## Feature Ranking Using $\mathrm{PG}^2$

We also study and evaluate a method for ranking the features of a model using the squared prediction gap called *greedy* $\mathrm{PG}^2$. Let $x \in \mathbb{R}^d$. Suppose the feature perturbations $\delta_i$ all come from $\mathcal{D} = \mathcal{N}(0, \sigma^2)$. The ranking $\pi^\sigma_{\mathrm{PG}}(x)$ is constructed in the following inductive manner.

For $l = 0, 1, \ldots, d-1$, assume we have already computed the $l$ top features $\pi^\sigma_{\mathrm{PG}}(x)(1), \ldots, \pi^\sigma_{\mathrm{PG}}(x)(l)$. Let $S = \{\pi^\sigma_{\mathrm{PG}}(x)(1), ..., \pi^\sigma_{\mathrm{PG}}(x)(l)\}$. For every $i \in [d] \setminus S$, compute $p_i := \mathrm{PG}^2(x, S \cup \{i\})$. We define the next important feature $\pi^\sigma_{\mathrm{PG}}(x)(l+1)$ to be the feature $i$ for which $p_i$ is the highest.

In the following, we compare the feature rankings obtained using the above method with the rankings obtained using the popular SHAP attribution framework (Lundberg et al. 2020) for tree ensembles, concretely using the `tree_path_dependent` feature perturbation method.

For a given $x \in \mathbb{R}^d$, SHAP produces an attribution vector $(\phi)_{i=1}^d$ whose elements sum up to $f(x) - \mathbb{E}[f(x)]$ but can be both positive and negative (Lundberg and Lee 2017). We define the ranking $\pi_{\mathrm{SHAP}}(x)$ to be obtained by sorting the features $[d]$ by the *absolute value* of $\phi_i$, from highest to lowest. This is justified by the fact that the predictions $f(x)$ can be smaller than the expected value; in such a case, all the attributions $\phi_i$ might be negative.

A natural question arises whether this ranking method produces rankings similar or dissimilar to SHAP rankings. Table 2 shows the ratio of datapoints for which the top $k$ most important features are the same for $\pi^\sigma_{\mathrm{PG}}$ and $\pi_{\mathrm{SHAP}}$. As we see, this ratio usually decreases with $k$, increases with $\sigma$, and is lower for the bigger models than for the single tree models. However, even the rankings for single tree models pick the same most important feature as SHAP less than 76% of the time. In general, the rankings produced are quite different, so it is worthwhile to study these differences and perform a comparative analysis of the rankings.

### Comparison vs SHAP wrt. $\mathrm{PGI}^2$

We perform a *global* comparison of $\pi^\sigma_{\mathrm{PG}}$ and $\pi_{\mathrm{SHAP}}$ wrt. the $\mathrm{PGI}^2$ metric, that is, we check which ranking method gives higher $\mathrm{PGI}^2$ scores on average. More formally, fix a dataset/model combination $m$ and a perturbation distribution $\mathcal{D}' = \mathcal{N}(0, (\sigma')^2)$, where potentially $\sigma' \neq \sigma$, for perturbations used in computing $\mathrm{PG}^2(x, \cdot)$ in (3). Let $X \subseteq \mathbb{R}^d$ be the corresponding dataset used in $m$. Then, for $\pi \in \{\pi_{\mathrm{SHAP}}, \pi^\sigma_{\mathrm{PGI}}\}$, define the average $\mathrm{PGI}^2$ scores as

$$\overline{\mathrm{PGI}^2}(\pi) = \frac{1}{|X|} \sum_{x \in X} \mathrm{PGI}^2(x, \pi(x)).$$

Table 3 presents the average scores $\overline{\mathrm{PGI}^2}(\pi_{\mathrm{SHAP}})$ and $\overline{\mathrm{PGI}^2}(\pi^\sigma_{\mathrm{PGI}})$ for $\sigma \in \{0.1, 0.3, 1.0\}$ as a function of $\sigma'$ for Bigger models (an analogous table for Single models can be found in the Appendix). We observe that even if the parameter $\sigma$ used when computing the ranking $\pi^\sigma_{\mathrm{PG}}$ differs from the parameter $\sigma'$ used for computing the $\mathrm{PGI}^2$ metrics, $\overline{\mathrm{PGI}^2}(\pi^\sigma_{\mathrm{PGI}})$ is almost always better than $\overline{\mathrm{PGI}^2}(\pi_{\mathrm{SHAP}})$

for the Bigger models, with a slight exception in the $\sigma' = 1.0, \sigma = 0.1$ pair case, when $\sigma, \sigma'$ differ a lot.

We conclude that the greedy optimization of the $\mathrm{PGI}^2(x, \pi)$ metric performs reasonably well, even though its correctness could only be guaranteed if it optimized the $\mathrm{PGI}(x, \{\pi_1\})$ metric (where only the single most important feature is perturbed) instead of (3).

## Comparison vs SHAP with Feature Removing

We compare the rankings $\pi_{\text{SHAP}}$ and $\pi^{\sigma}_{\mathrm{PG}}$ using feature removing. Intuitively, removing information about the value of a highly important feature should result in greater model error than removing information about the value of a less important feature. Thus, we can assess a ranking by measuring the error of the model when the features the ranking deems most important are removed. More precisely, for a ranking $\pi$ and for each $k = 1, 2, ...$ we measure the RMSE of $\xi(x, [d] \backslash \pi(x)[1..k])$ over all $x$ from the test dataset, where $\xi(\cdot, S)$ is a function giving model predictions based only on features from $S$. We have two such functions $\xi$ (and thus two different methods of comparing rankings) defined below.

The first method is feature randomising. We define $\xi_{random}(x, S) := \mathbb{E}[m(X)]$ where $m$ is the model and $X$ is a random variable with $X_i \equiv x_i$ for $i \in S$ and $X_j$ has the distribution of the $j$-th feature over the test set for $j \notin S$. In practice, the expected value is approximated by sampling $m(X)$ 100 times (where each feature is sampled independently) and taking the average.

| $\sigma$ | $k=1$ | $k=2$ | $k=3$ |
|---|---|---|---|
| Single tree Red Wine Quality model | | | |
| 0.1 | 0.328 | 0.259 | 0.031 |
| 0.3 | 0.412 | 0.591 | 0.428 |
| 1.0 | 0.419 | 0.613 | 0.506 |
| Bigger Red Wine Quality model | | | |
| 0.1 | 0.131 | 0.041 | 0.022 |
| 0.3 | 0.212 | 0.094 | 0.056 |
| 1.0 | 0.391 | 0.181 | 0.103 |
| Single California Housing model | | | |
| 0.1 | 0.501 | 0.263 | 0.031 |
| 0.3 | 0.599 | 0.569 | 0.349 |
| 1.0 | 0.712 | 0.471 | 0.504 |
| Bigger California Housing model | | | |
| 0.1 | 0.194 | 0.08 | 0.052 |
| 0.3 | 0.3 | 0.122 | 0.056 |
| 1.0 | 0.569 | 0.243 | 0.08 |
| Single Parkinson Telemonitoring model | | | |
| 0.1 | 0.402 | 0.143 | 0.066 |
| 0.3 | 0.569 | 0.32 | 0.237 |
| 1.0 | 0.759 | 0.414 | 0.22 |
| Bigger Parkinson Telemonitoring model | | | |
| 0.1 | 0.137 | 0.028 | 0.007 |
| 0.3 | 0.225 | 0.061 | 0.009 |
| 1.0 | 0.368 | 0.114 | 0.031 |

Table 2: Ratio of datapoints having the same $k$ most important features in $\pi^{\sigma}_{\mathrm{PG}}$ as in $\pi_{\text{SHAP}}$

| $\sigma'$ | SHAP | $\sigma = 0.1$ | $\sigma = 0.3$ | $\sigma = 1.0$ |
|---|---|---|---|---|
| Bigger PGI Red Wine Quality model | | | | |
| 0.1 | 0.013 | 0.024 | 0.022 | 0.018 |
| 0.3 | 0.051 | 0.068 | 0.078 | 0.067 |
| 1.0 | 0.221 | 0.192 | 0.266 | 0.302 |
| Bigger PGI California Housing model | | | | |
| 0.1 | 3.346e+08 | 4.205e+08 | 3.841e+08 | 3.065e+08 |
| 0.3 | 1.209e+09 | 1.297e+09 | 1.374e+09 | 1.259e+09 |
| 1.0 | 7.807e+09 | 6.963e+09 | 8.266e+09 | 8.728e+09 |
| Bigger PGI Parkinson Telemonitoring model | | | | |
| 0.1 | 2.743 | 4.425 | 4.015 | 3.199 |
| 0.3 | 6.341 | 9.458 | 11.878 | 10.143 |
| 1.0 | 16.298 | 19.56 | 30.188 | 34.013 |

Table 3: Greedy $\mathrm{PG}^2$ vs SHAP rankings wrt. $\mathrm{PGI}^2$.

The other method follows the remove-and-retrain approach described in (Hooker et al. 2019). For $k = 1, 2, 3, ...$ and for each subset of features $S \subset [d]$ such that $|S| = d - k$ we train a XGBoost model where we only use the features in $S$. That is, the $k$ features that are not in $S$ are not available for the training of the model. This way, we obtain a family of models $(m_S)_{S \subset [d]}$ where each model $m_S$ gives predictions based only on values of features in $S$. We define $\xi_{retrain}(x, S) := m_S(x)$.

The comparison was performed for $k = 1, 2, 3$. The results are shown in Tables in the Appendix. The results for the retraining method suggest that $\pi_{\text{SHAP}}$ picks the most important features more accurately than $\pi^{\sigma}_{\mathrm{PG}}$. The results for the randomising method show no significant difference between $\pi_{\text{SHAP}}$ and $\pi^{\sigma}_{\mathrm{PG}}$ for Red Wine Quality and Parkinson Telemonitoring models. However, for the Housing model we see that $\pi^{\sigma}_{\mathrm{PG}}$ with $\sigma \in \{0.1, 0.3\}$ give better results than $\pi_{\text{SHAP}}$. In general, this comparison doesn't give a simple conclusion that one ranking method is clearly better than the other. Each of them can show advantage over the other depending on the dataset and assessment method.

## Conclusion

In this work, we proposed a cleanly defined faithfulness metric $\mathrm{PGI}^2$ that can be computed exactly in polynomial time on tree ensemble models. We designed a quadratic algorithm for that. Our experimental evaluation showed that the algorithm is numerically stable and superior to Monte Carlo-based methods if very accurate results are desired.

We also proposed a natural feature ranking method inspired by $\mathrm{PGI}^2$ optimization. The method generally identifies different most important features than the state-of-the-art SHAP explainer while offering similar performance wrt. different faithfulness metrics. The $\mathrm{PG}^2$-based ranking may thus offer a viable alternative for assessing feature importance in tree-based models.

## Acknowledgments

Partially supported by the ERC PoC grant EXALT no 101082299 and the National Science Centre (NCN) grant no. 2020/37/B/ST6/04179.

## References


Agarwal, C.; Krishna, S.; Saxena, E.; Pawelczyk, M.; Johnson, N.; Puri, I.; Zitnik, M.; and Lakkaraju, H. 2022. Openxai: Towards a transparent evaluation of model explanations. *Advances in Neural Information Processing Systems*, 35: 15784–15799.

Alvarez-Melis, D.; and Jaakkola, T. S. 2018. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*.

Balagopalan, A.; Zhang, H.; Hamidieh, K.; Hartvigsen, T.; Rudzicz, F.; and Ghassemi, M. 2022. The road to explainability is paved with bias: Measuring the fairness of explanations. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 1194–1206.

Burkart, N.; and Huber, M. F. 2021. A Survey on the Explainability of Supervised Machine Learning. *J. Artif. Intell. Res.*, 70: 245–317.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM.

Cortez, P.; Cerdeira, A.; Almeida, F.; Matos, T.; and Reis, J. 2009. Wine Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56S3T.

Dai, J.; Upadhyay, S.; Aivodji, U.; Bach, S. H.; and Lakkaraju, H. 2022. Fairness via explanation quality: Evaluating disparities in the quality of post hoc explanations. In *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, 203–214.

DeYoung, J.; Jain, S.; Rajani, N. F.; Lehman, E.; Xiong, C.; Socher, R.; and Wallace, B. C. 2020. ERASER: A Benchmark to Evaluate Rationalized NLP Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 4443–4458. Association for Computational Linguistics.

Hooker, S.; Erhan, D.; Kindermans, P.; and Kim, B. 2019. A Benchmark for Interpretability Methods in Deep Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 9734–9745.

Liu, Y.; Khandagale, S.; White, C.; and Neiswanger, W. 2021. Synthetic Benchmarks for Scientific Research in Explainable Machine Learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Lundberg, S. M.; Erion, G. G.; Chen, H.; DeGrave, A. J.; Prutkin, J. M.; Nair, B.; Katz, R.; Himmelfarb, J.; Bansal, N.; and Lee, S. 2020. From local explanations to global understanding with explainable AI for trees. *Nat. Mach. Intell.*, 2(1): 56–67.

Lundberg, S. M.; and Lee, S.-I. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Petsiuk, V.; Das, A.; and Saenko, K. 2018. RISE: Randomized Input Sampling for Explanation of Black-box Models. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, 151. BMVA Press.

Plumb, G.; Molitor, D.; and Talwalkar, A. 2018. Model Agnostic Supervised Local Explanations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2520–2529.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 1135–1144. ACM.

Torgo, L. 2023. California Housing Dataset. https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html. Accessed on: October 10, 2023.

Tsanas, A.; and Little, M. 2009. Parkinsons Telemonitoring. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5ZS3N.

Yeh, C.-K.; Hsieh, C.-Y.; Suggala, A.; Inouye, D. I.; and Ravikumar, P. K. 2019. On the (in) fidelity and sensitivity of explanations. *Advances in Neural Information Processing Systems*, 32.