Statistics and Computing (2024) 34:149
https://doi.org/10.1007/s11222-024-10459-9

ORIGINAL PAPER

Efficient Shapley performance attribution for least-squares regression

Logan Bell[1] · **Nikhil Devanathan**[2] · **Stephen Boyd**[2]

Received: 1 February 2024 / Accepted: 18 June 2024

Abstract
We consider the performance of a least-squares regression model, as judged by out-of-sample R^2. Shapley values give a fair attribution of the performance of a model to its input features, taking into account interdependencies between features. Evaluating the Shapley values exactly requires solving a number of regression problems that is exponential in the number of features, so a Monte Carlo-type approximation is typically used. We focus on the special case of least-squares regression models, where several tricks can be used to compute and evaluate regression models efficiently. These tricks give a substantial speed up, allowing many more Monte Carlo samples to be evaluated, achieving better accuracy. We refer to our method as least-squares Shapley performance attribution (LS-SPA), and describe our open-source implementation.

Keywords Feature importance · Least squares · Monte Carlo

1 Introduction

We consider classic least-squares regression, with p features, judged by an out-of-sample R^2 metric. A natural question is how much each of the p features contributes to our R^2 metric; roughly speaking, how valuable is each feature to our least-squares predictor? Except for a special case described below in Sect. 2.4, this question seems difficult to answer, since the value of a feature depends on the other features.

Our interest is in attributing the *overall performance* of a least-squares model to the features. A related task is attributing a *specific prediction* of a least-squares model to the features, which is a popular method for so-called explainable AI called SHAP, an acronym for Shapley additive explanations (Lundberg and Lee 2017; Molnar 2022; Chen et al. 2023). That is a very different task, discussed in more detail below. In this paper, we consider only performance attribution, and not explaining a specific prediction from a model. We refer to this task as Shapley performance attribution to features.

This performance attribution problem was essentially solved in Lloyd Shapley's 1953 paper "A Value for n-Person Games" (Shapley 1952). He proposed a method to allocate the payoff in a cooperative game to the players, which came to be known as Shapley values. Shapley values provide a fair distribution of the total payoff in a game, taking into account the contributions of each player to the coalition. Shapley values are provably the only attribution for which fairness, monotonicity, and full attribution (three key desiderata for attribution) all hold. We refer the reader to other papers for more discussion and justification of Shapley values for attributing regression model performance to its features (Huettner and Sunder 2012; Zhang et al. 2023; Fryer et al. 2021; Owen and Prieur 2017).

We focus on efficiently computing (an approximation of) the Shapley values for least-squares regression problems, i.e., to attribute the overall R^2 to the p features. We seek a number S_j associated with feature j, where we interpret S_j as the portion of the achieved R^2 metric that is attributed to feature j. Full attribution means $\sum_{j=1}^{p} S_j = R^2$.

Shapley values rely on solving and evaluating around 2^p least-squares problems. This is impractical for p larger than around 10, so Monte Carlo approximation is typically used to compute an approximation to the Shapley values. We propose a simple but effective quasi-Monte Carlo method that

✉ Logan Bell
lmbell@alumni.stanford.edu

Nikhil Devanathan
ndevanathan@alumni.stanford.edu

Stephen Boyd
boyd@stanford.edu

[1] Computational and Mathematical Engineering, Stanford University, 475 Via Ortega, Suite B060, Stanford, CA 94305-4042, USA

[2] Electrical Engineering, Stanford University, 350 Serra Mall, Stanford, CA 94305-1234, USA

Published online: 04 July 2024

in practice gives better approximations of the Shapley values than Monte Carlo for the same number of least-square regression problems.

We do not introduce any new mathematical or computational methods. Instead, we collect well-known ideas and assemble them into an efficient method for computing the Shapley values for a least-squares regression problem, exploiting special properties of least-squares problems.

1.1 Prior work

1.1.1 Cooperative game theory

Shapley values originated in cooperative game theory as a means of fairly splitting a coalition's reward between the individual players (Shapley 1952). The notion of a fair split is defined by four axioms, which Shapley proved resulted in a unique method for attribution. Since Shapley's seminal paper, numerous extensions, variations, and generalizations have been developed; see, for instance, Monderer and Samet (2002), Dubey et al. (1981), Owen (1977), Algaba et al. (2019), Chalkiadakis et al. (2012), Kóczy (2007).

Computing the Shapley values in general has a cost that increases exponentially in the number of players. Nonetheless, many games have structure that enables efficient exact computation of the Shapley values. Examples include weighted hypergraph games with fixed coalition sizes (Deng and Papadimitriou 1994), determining airport landing costs (Littlechild and Owen 1973), weighted voting games restricted by trees (Fernández et al. 2002), cost allocation problems framed as extended tree games (Granot et al. 2002), sequencing games (Curiel et al. 1989), games represented as marginal contribution networks (Ieong and Shoham 2005), and determining certain notions of graph centrality (Michalak et al. 2013). On the other hand, computing Shapley values in weighted majority games is #P-complete (Deng and Papadimitriou 1994), as are elementary games, i.e., games whose value function is an indicator on a coalition (Faigle and Kern 1992).

1.1.2 Approximating Shapley values

Due to the computational complexity of computing exact Shapley values in general, various methods have been proposed for efficiently approximating Shapley values. Shapley initially described a Monte Carlo method for approximating Shapley values by sampling coalitions in 1960 (Mann and Shapley 1960). Subsequent works have considered sampling permutations using simple Monte Carlo methods (Zlotkin and Rosenschein 1994; Castro et al. 2009; Moehle et al. 2022), stratified and quasi-Monte Carlo methods (Campen et al. 2017; Castro et al. 2017; Maleki et al. 2014; Mitchell et al. 2022), and ergodic sampling methods (Illés and Kerényi 2022).

Beyond Monte Carlo approaches, other works have explored numerical integration schemes for approximating the Shapley values. The paper (Owen 1972) describes a multilinear extension of the characteristic function of an n-person game that allows for the computation of the Shapley values as a contour integral. This method has been further explored in Leech (2003) and Fatima et al. (2008).

1.1.3 Applications of Shapley values

Although they arose in the context of game theory, Shapley values have been applied across a variety of fields. In finance, Shapley values have been applied to attribute the performance of a portfolio to constituent assets (Moehle et al. 2022) and to allocate insurance risk (Powers 2007). Elsewhere, Shapley values have been used to identify key individuals in social networks (Michalak et al. 2013; Campen et al. 2017), to identify which components of a user interface draw the most user engagement (Zhao et al. 2018), to distribute rewards in multi-agent reinforcement learning (Wang et al. 2020), and to attribute the performance of a machine-learning model to the individual training data points (Ghorbani and Zou 2019). We refer to Moretti and Patrone (2008) and Algaba et al. (2019) for a deeper review of applications of Shapley values.

1.1.4 Explainable ML

Shapley attribution has recently found extensive use in machine learning in the context of model interpretability, in Shapley additive explanation (SHAP) (Lundberg and Lee 2017). SHAP uses approximate Shapley values to attribute a single prediction of a machine-learning model across the input features. Although SHAP and Shapley performance attribution both involve prediction models and both use Shapley values, they otherwise have little relation. We refer to Molnar (2022) and Chen et al. (2023) for a more thorough review of SHAP.

1.1.5 Shapley values for statistics

In statistical learning, researchers often seek to assign a relative importance score to the features of a model. One approach is Shapley attribution. This method has been independently rediscovered numerous times and called numerous names (Lindeman et al. 1980; Lipovetsky and Conklin 2001; Kruskal 1987; Mishra 2016; Grömping 2006, 2015). All of these works utilize Shapley attribution to decompose the R^2 of a regression model, though often without reference to Shapley. The paper (Budescu 1993) decomposes the R^2 using a method similar to Shapley attribution but with different weights, and Chevan and Sutherland (1991) decomposes any

goodness-of-fit metric of a regression model using a method shown in Stufken (1992) to be equivalent to Shapley attribution.

While not directly related to the computation of Shapley values, the application of Shapley values to feature importance is a primary motivation behind their calculation in many contexts (Moehle et al. 2022; Michalak et al. 2013; Campen et al. 2017). In statistics, the use of Shapley values for determining feature importance has been significantly explored (Kumar et al. 2020; Harris et al. 2022; Williamson and Feng 2020; Fryer et al. 2021; Owen and Prieur 2017), and papers (Huettner and Sunder 2012; Zhang et al. 2023; Fryer et al. 2021; Owen and Prieur 2017) further argue why Shapley attribution is a particularly appropriate method for evaluating feature importance.

1.2 This paper

We introduce an efficient method for approximating Shapley attribution of performance in least-squares regression problems, called least-squares Shapley performance attribution (LS-SPA). LS-SPA uses several computational tricks that exploit special properties of least-squares problems. The first is a reduction of the original train and test data to a compressed form in which the train and test data matrices are square. The second is to solve a set of p least-squares problems, obtained as we add features one by one, with one QR factorization, in a time comparable to solving one least-squares problem. Finally, we propose using a quasi-Monte Carlo method, a variation of Monte Carlo sampling, to efficiently approximate the Shapley values. (This trick does not depend on any special properties of least-squares problems.)

1.2.1 Outline

In Sect. 2 we present a mathematical overview of least-squares and Shapley values, setting our notation. We describe our method for efficiently estimating Shapley values for least-squares problems in Sect. 3. In Sect. 4, we describe some extensions and variations on our algorithm, and we conclude with numerical experiments in Sect. 5.

2 Least-squares Shapley performance values

In this section, we review the least-squares regression problem, set our notation, and define the Shapley values for the features.

2.1 Least-squares

We consider the least-squares regression problem

$$\text{minimize } \|X\theta - y\|_2^2, \tag{1}$$

with variable $\theta \in \mathbf{R}^p$, the model parameter. Here $X \in \mathbf{R}^{N\times p}$ is a given data or feature matrix and $y \in \mathbf{R}^N$ is a given vector of responses or labels. The rows of X, denoted x_i^T with $x_i \in \mathbf{R}^p$, correspond to N samples or observations, and each column of X corresponds to a feature. We will assume that X has rank p, which implies $N \geq p$, *i.e.*, X is square or tall. We denote the solution of the least-squares problem (1) as

$$\theta^\star = X^\dagger y = (X^T X)^{-1} X^T y.$$

The data X and y are the training data since they are used to find the model parameter $\theta^\star$.

The least-squares problem (1) yields a linear model $\hat{y} = x^T\theta$ with $\theta = \theta^\star$. We can include a constant offset or intercept in the model, $\hat{y} = x^T\theta + \beta$, several ways. One method is to include a feature that has the constant value one, so the formulation above (1) is unchanged. In this case, however, our attribution gives an attribution to the offset, which might not be wanted. Another method is to solve (1) with centered data, *i.e.*, data with the average of each feature, and the labels, subtracted, so they all have zero mean. To see this, note that the optimization problem

$$\text{minimize } \|X\theta + \beta\mathbf{1} - y\|_2^2$$

with variables $\theta \in \mathbf{R}^p$ and $\beta \in \mathbf{R}$ has optimal variables

$$\theta^\star = (X - \mathbf{1}\bar{x})^\dagger(y - \mathbf{1}\bar{y}), \quad \beta^\star = \frac{1}{N}\mathbf{1}^T(y - X\theta^\star).$$

Here, $\bar{x} = \frac{1}{N}\mathbf{1}^T X$ is the sample mean of the feature vectors and $\bar{y} = \frac{1}{N}\mathbf{1}^T y$ is the sample mean of the labels. Hence, to fit a model with an intercept, we can solve (1) with centered data to obtain $\theta^\star$, and from this recover $\beta^\star$. In this formulation, we do not attribute performance to the offset constant. In the sequel we do not include the intercept, noting that an offset can be included by centering the data.

2.1.1 Out-of-sample R^2 metric

We evaluate the performance of a model parameter θ via out-of-sample validation. We have a second (test) data set of M observations $X^{\text{tst}} \in \mathbf{R}^{M\times p}$ and $y^{\text{tst}} \in \mathbf{R}^M$ and evaluate the model on these data to obtain $\hat{y}^{\text{tst}} = X^{\text{tst}}\theta$. The prediction errors on the test set are given by $\hat{y}^{\text{tst}} - y^{\text{tst}}$. To evaluate the

least-squares model with parameter θ, we use the R^2 metric

$$R^2 = \frac{\|y^{\text{tst}}\|_2^2 - \|\hat{y}^{\text{tst}} - y^{\text{tst}}\|_2^2}{\|y^{\text{tst}}\|_2^2}, \tag{2}$$

which is the fractional reduction in mean square test error compared to the baseline prediction $\hat{y} = 0$. Larger values of R^2 are better. It is at most one and can be negative.

In this paper, we focus exclusively on the out-of-sample R^2 metric to evaluate a least-squares model. However, the algorithm we develop and present in Sect. 3 can be used to attribute any in-sample or out-of-sample performance metric across the features of a least-squares model.

2.2 Feature subsets and chains

2.2.1 Feature subsets

In later sections, we will be interested in the R^2 metric obtained with the least-squares model using only a subset $\mathcal{S} \subseteq \{1, \ldots, p\}$ of the features, *i.e.*, using a parameter vector θ that satisfies $\theta_j = 0$ for $j \notin \mathcal{S}$. The associated least-squares problem is

$$\begin{array}{ll} \text{minimize} & \|X\theta - y\|_2^2 \\ \text{subject to} & \theta_j = 0, \quad j \notin \mathcal{S}. \end{array} \tag{3}$$

We denote the associated parameter as $\theta^\star_{\mathcal{S}}$. From this we can find the R^2 metric, denoted $R^2_{\mathcal{S}}$, using (2). We use R^2 to denote the metric obtained using all features, *i.e.*, $R^2_{\{1,\ldots,p\}}$.

2.2.2 Feature chains

A *feature chain* is an increasing sequence of p subsets of features obtained by adding one feature at a time,

$$\emptyset \subset \mathcal{S}_1 \subset \cdots \subset \mathcal{S}_p = \{1, \ldots, p\},$$

where $|\mathcal{S}_k| = k$. We denote π_k as the index of the feature added to form $\mathcal{S}_k$. Evidently $\pi = (\pi_1, \ldots, \pi_p)$ is a permutation of $(1, \ldots, p)$. With this notation we have

$$\mathcal{S}_k = (\pi_1, \ldots, \pi_k), \quad k = 1, \ldots, p.$$

Roughly speaking, π gives the order in which we add features in the feature chain. We will set $\mathcal{S}_0 = \emptyset$.

2.2.3 Lifts associated with a feature chain

Consider feature j. It is the lth feature to be added in the feature chain given by π, where $l = \pi^{-1}(j)$. We define the *lift* associated with feature j in chain π as

$$L(\pi)_j = R^2_{\mathcal{S}_l} - R^2_{\mathcal{S}_{l-1}}.$$

Roughly speaking, $L(\pi)_j$ is the increase in R^2 obtained when we add feature j to the ones before it in the ordering π, *i.e.*, features $\pi_1, \ldots, \pi_{l-1}$. The lift $L(\pi)_j$ can be negative, which means that adding feature j to the ones that come before it reduces the R^2 metric.

We refer to the vector $L(\pi) \in \mathbf{R}^p$ as the lift vector associated with the feature chain given by π. We observe that

$$\sum_{j=1}^p L(\pi)_j = \sum_{j=1}^p \left(R^2_{\mathcal{S}_l} - R^2_{\mathcal{S}_{l-1}} \right) = R^2,$$

the R^2 metric obtained using all features. The vector $L(\pi)$ gives an attribution of the values of each feature to the final R^2 obtained, assuming the features are added in the order π. In general, it depends on π.

2.3 Shapley attributions

The vector of Shapley attributions for the features, denoted $S \in \mathbf{R}^p$, is given by

$$S = \frac{1}{p!} \sum_{\pi \in \mathcal{P}} L(\pi), \tag{4}$$

where $\mathcal{P}$ is the set of all $p!$ permutations of $\{1, \ldots, p\}$. We interpret S_j as the average lift, or increase in R^2, obtained when adding feature j over all feature chains. The average is over all feature chains, *i.e.*, orderings of the features. In Sect. 2.5, we present a simple example of a Shapley attribution for a least-squares model with a small number of features.

For p more than 10 or so, it is impractical to evaluate the lift vector for all $p!$ permutations. Instead, we estimate it as

$$\hat{S} = \frac{1}{K} \sum_{\pi \in \Pi} L(\pi), \tag{5}$$

where $\Pi \subset \mathcal{P}$ is a subset of permutations with $|\Pi| = K \ll p!$. This is a Monte Carlo approximation of (4) when Π is a subset of permutations chosen uniformly at random from $\mathcal{S}$ with replacement. (We will describe a better choice in Sect. 3.5.)

2.4 Uncorrelated features

We mention here one case in which the Shapley performance attribution for least-squares regression is easily found: When the empirical covariance of the features on both the train and test sets are diagonal, *i.e.*,

$$(1/N)X^T X = \Lambda, \qquad (1/M)(X^{\text{tst}})^T X^{\text{tst}} = \tilde{\Lambda},$$

Table 1 R^2 for each subset $\mathcal{S}$ of the features

$\mathcal{S}$	R^2
{1, 2, 3}	0.92
{1, 2}	0.92
{1, 3}	0.82
{2, 3}	0.69
{1}	0.81
{2}	0.69
{3}	−0.43
∅	0.00

with Λ and $\tilde{\Lambda}$ diagonal. In this case, we have $\theta_j^\star = \Lambda_{jj}^{-1}(X^T y)_j$, for any subset $\mathcal{S}$ that contains j. The test error is also additive, *i.e.*, the sum of contributions from each feature. It follows that the lift vectors do not depend on π, so $S = L(\pi)$ for any π.

When these assumptions almost hold, *i.e.*, the features are not too correlated on the train and test sets, the method we propose exhibits very fast convergence.

2.5 Toy example

To illustrate the ideas above we present a simple example. We use a synthetic dataset with $p = 3$ features, $N = 50$ training examples, and $M = 50$ test examples. We generate feature matrices X and X^{tst} by taking, respectively, N and M independent samples from a multivariate normal distribution with mean zero and covariance

$$\Sigma = \begin{bmatrix} 1.0 & 0.7 & -0.4 \\ 0.7 & 1.0 & -0.5 \\ -0.4 & -0.5 & 1.0 \end{bmatrix}.$$

Using true weights $\theta = (2.1, 1.4, 0.1)$, we take $y = X\theta + \omega$ and $y^{\text{tst}} = X^{\text{tst}}\theta + \omega^{\text{tst}}$ where the entries of $\omega \in \mathbf{R}^N$ and $\omega^{\text{tst}} \in \mathbf{R}^M$ are independently sampled from a standard normal distribution.

Table 1 shows the out-of-sample R^2 for each of the 8 subsets of features. Table 2 shows the lift associated with each of 6 feature orderings. We display the same data as a lattice in Fig. 1. In this figure vertices are labeled with subsets of the features and subscripted with the associated R^2. The edges, oriented to point to the subset to which one feature was added, are labeled with the lift for adding that feature to the subset. Every path from ∅ to {1, 2, 3} corresponds to an ordering of the features, with the lifts along the path giving the associated lift vector.

The R^2 using all features is 0.92, and the Shapley values are

$$S = (0.59, 0.47, -0.14).$$

Table 2 Lift vector L generated by each permutation π of the features

π	$L(\pi)$		
(1, 2, 3)	(0.81,	0.11,	0.00)
(1, 3, 2)	(0.81,	0.10,	0.01)
(2, 1, 3)	(0.23,	0.69,	0.00)
(2, 3, 1)	(0.23,	0.69,	0.00)
(3, 1, 2)	(1.25,	0.10,	−0.43)
(3, 2, 1)	(0.23,	1.12,	−0.43)

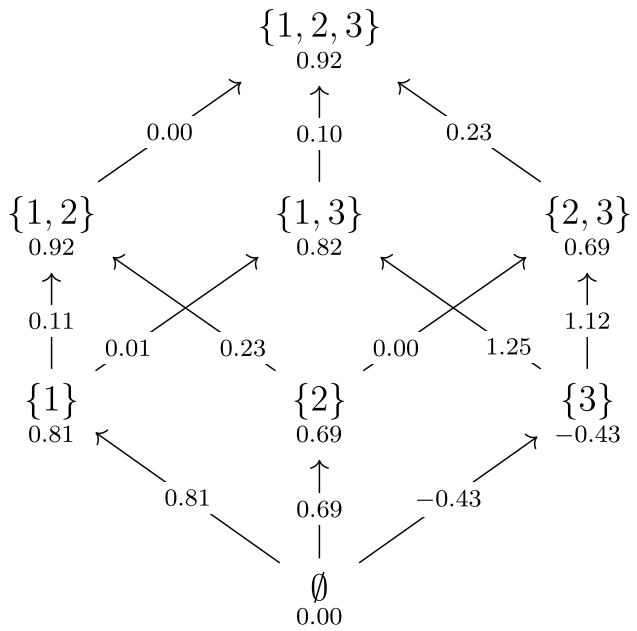

Fig. 1 Shapley attribution on the toy data represented as a lattice

Roughly speaking, most of our performance comes from feature 1, followed closely by feature 2, with feature 3 negatively affecting performance. (Since R^2 is evaluated out of sample, it can be negative.) Indeed, we can see that the performance using only features 1 and 2 is the same, to two decimal places, as the performance using all three.

3 Efficient computation

In this section we explain LS-SPA, our method for efficiently computing $\hat{S}$, an approximation of S. The method can be broken into two parts. The first is a method to efficiently compute $L(\pi)$, the lift associated with a specific feature ordering π. The second is a method for choosing the set of permutations Π that gives a better approximation than basic Monte Carlo sampling.

3.1 The naïve method

The naïve method for computing $\hat{S}$ is to solve a chain of p least-squares problems K times, and evaluate them on a test set. Solving a least-squares problem with k (nonzero) coefficients has a cost $O(Nk^2)$ flops. (It can be done, for example, via the QR factorization.) Evaluating its performance costs $O(Mk)$. Assuming M is no more than Nk in order, this second term is negligible. Summing $O(Nk^2)$ from $k = 1$ to p gives $O(Np^3)$. This is done for K permutations so the naïve method requires

$$O(KNp^3) \tag{6}$$

flops. This naïve method can be parallelized: All of the least-squares problems can be solved in parallel.

We will describe a method to carry out this computation far more efficiently. The computation tricks we describe below are all individually well known; we are merely assembling them into an efficient method.

3.2 Initial reduction of training and test data sets

We can carry out an initial reduction of the original train and test data matrices, so each has p rows instead of N and M respectively. Let $X = QR$ denote the QR factorization of X, with $Q \in \mathbf{R}^{N\times p}$ and $R \in \mathbf{R}^{p\times p}$. Simple algebra shows that

$$\|X\theta - y\|_2^2 = \|R\theta - Q^T y\|_2^2 + \|y - Q(Q^T y)\|_2^2. \tag{7}$$

The righthand side consists of a least-squares objective with square data matrix R and righthand side $\tilde{y} = Q^T y$, plus a constant. The cost to compute R and $\tilde{y} = Q^T y$ is $O(Np^2)$. We do this once and then solve the least-squares problem (3) using the objective $\|R\theta - \tilde{y}\|_2^2$. The cost for this is $O(pk^2)$, where $k = |\mathcal{S}|$.

Computing the least-squares solutions for a chain now costs $O(p^4)$, whereas in the naïve method, the cost was $O(Np^3)$ per chain. The cost of computing least-squares solutions for K chains is then

$$O(Np^2 + Kp^4),$$

compared to $O(KNp^3)$ for the naïve method. When N or K is large (which is typical), the cost savings are substantial.

The same trick can be used to efficiently evaluate the R^2 metrics. We carry out one QR factorization of the test matrix at a cost of $O(Mp^2)$, after which we can evaluate the metric with $O(pk)$ flops, where $k = |\mathcal{S}|$. To evaluate the metrics for a chain is then $O(p^2)$ flops, compared to $O(Mp)$ for the naïve method. To compute $\hat{S}$ for K chains has cost

$$O(Mp^2 + Kp^2),$$

which is negligible compared to the cost of solving the least-squares problems.

Using this initial reduction trick, we obtain a complexity of $O(Np^2 + Kp^4)$, compared to $O(KNp^3)$ for the naïve method. This simple trick has been known since at least the 1960s (Businger and Golub 1965; Golub 1965). A similar reduction trick uses a Cholesky factorization, rather than a QR factorization, which is useful when the data set is too large to fit in memory; see Sect. 4.3 for more details.

3.3 Efficiently computing lift vectors

In this section, we show how the cost of computing lift vectors and evaluating them for one chain can be reduced from $O(p^4)$ to $O(p^3)$, using a well-known property of the QR factorization.

Given a feature chain $\mathcal{S}_0 \subset \mathcal{S}_1 \subset \cdots \subset \mathcal{S}_p$ associated with a permutation π, solving (3) for $\mathcal{S}_i$ is equivalent to solving

$$\begin{array}{l}\text{minimize } \|RP_\pi^T\tilde{\theta} - \tilde{y}\|_2^2 \\ \text{subject to } \tilde{\theta}_j = 0, \quad j > i,\end{array} \tag{8}$$

with variable $\tilde{\theta} \in \mathbf{R}^p$. Here R and $\tilde{y} = Q^T y$ are the reduced data obtained from Sect. 3.2 and P_π is the permutation matrix associated with π. The kth column of RP_π^T is the π_kth column of R. The optimal parameter $\theta^\star$ of (3) is related to the optimal parameter $\tilde{\theta}^\star$ of (8) via $\theta^\star = P_\pi^T\tilde{\theta}^\star$.

We can combine the problems (8) for $i = 1, \ldots, p$ into one problem by collecting the parameter vectors $\tilde{\theta}$ into one $p \times p$ upper triangular matrix $\tilde{\Theta}$. We then solve

$$\begin{array}{l}\text{minimize } \|RP_\pi^T\tilde{\Theta} - \tilde{Y}\|_F^2 \\ \text{subject to } \tilde{\Theta} \text{ upper triangular},\end{array}$$

with variable $\tilde{\Theta} \in \mathbf{R}^{p\times p}$. Here $\|\cdot\|_F^2$ is the Frobenius norm squared, *i.e.*, the sum of the entries. The matrix $\tilde{Y}$ is given by $\tilde{Y} = \tilde{y}\mathbf{1}^T$, where $\mathbf{1}$ is the vector with all entries one, *i.e.*, $\tilde{Y}$ is the matrix with all columns $\tilde{y}$. (The p different least-squares problems are uncoupled, but it is convenient to represent them as one matrix least-squares problem (Boyd and Vandenberghe 2018).)

Let $\tilde{Q}\tilde{R} = RP_\pi^T$ denote the QR decomposition of RP_π^T. Substituting $\tilde{Q}\tilde{R}$ for RP_π^T above, and multiplying the argument of the Frobenius norm the orthogonal matrix $\tilde{Q}^T$, the problem above can be written as

$$\begin{array}{l}\text{minimize } \|\tilde{R}\tilde{\Theta} - \tilde{Q}^T\tilde{Y}\|_F^2 \\ \text{subject to } \tilde{\Theta} \text{ upper triangular},\end{array}$$

with variable $\tilde{\Theta} \in \mathbf{R}^{p\times p}$. The solution has the simple form

$$\tilde{\Theta}^\star = \tilde{R}^{-1}\mathbf{triu}(\tilde{Q}^T\tilde{Y}). \tag{9}$$

where **triu**$(\cdot)$ gives the upper triangular part of its argument, *i.e.*, sets the strictly lower triangular entries to zero. Note that the righthand side is upper triangular since upper triangularity is preserved under inversion and matrix multiplication. This result is equivalent to application of the Frish–Waugh–Lovell theorem from econometrics (Frisch and Waugh 1933; Lovell 1963) and is also well-known in statistics (Hastie et al. 2009). The optimal parameters for $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_p$ are thus the columns of $\Theta^\star = P_\pi^T \tilde{\Theta}^\star$

3.3.1 Complexity

Computing the QR factorization of RP_π^T costs $O(p^3)$. We can form $\tilde{Q}^T\tilde{Y} = \tilde{Q}^T\tilde{y}\mathbf{1}$ in $O(p^2)$, which is negligible. We can compute $\Theta^\star$ using (9) in $O(p^3)$ flops. In other words: We can find the parameter vectors for a whole chain in $O(p^3)$, the same cost as solving a single least-squares problem with p variables and p equations. We evidently save a factor of p, compared to the naïve method of solving p least-squares problems, which has cost $O(p^4)$.

It is easily verified that the cost of evaluating the p least-squares parameters on the test data is also $O(p^3)$, so the cost of evaluating the lifts for the chain is $O(p^3)$.

3.4 Summary

Altogether, the complexity of LS-SPA is

$$O(Np^2 + Kp^3), \tag{10}$$

which can be compared to the complexity of the naïve method, $O(KNp^3)$ (6). The speedup over the naïve method is at least the minimum of N and Kp, neither of which is typically small. We note that LS-SPA can also be parallelized, by computing the lifts for each $\pi \in \Pi$ in parallel.

3.5 Quasi-Monte Carlo approximation

Here we explain an improvement over the simple Monte Carlo method in (5). (This improvement has nothing to do with the problems being least-squares and is applicable in other cases.) We will use quasi-Monte Carlo (QMC) sampling instead of randomly sampling permutations to obtain Π. One proposed method (which we call *permutohedron QMC*) is given in Mitchell et al. (2022). It maps a Sobol' sequence in $[0, 1]^{p-2}$ onto the permutohedron for p-element permutations by mapping to the $(p-1)$-sphere, then embedding the $(p-1)$-sphere into $\mathbf{R}^p$ via an area-preserving transform and rounding points to the nearest permutohedron vertex.

We propose another method (which we call *argsort QMC*), which is to take a Sobol' sequence on $[0, 1]^p \subset \mathbf{R}^p$, and choose the permutations as the argsort (permutation that gives the sorted ordering) of each point in the sequence. We have found empirically that this method does as well as permutohedron sampling for this problem, and is computationally simpler.

We note that other quasi-Monte Carlo sequences, such as Halton sequences, may be used in place of Sobol' sequences. It is common to randomize quasi-Monte Carlo sequences as doing so can improve convergence rates, and some theoretical work exists to justify error estimation in this setting (Owen 1998, 2023). In our empirical studies, we have found scrambled Sobol' sequences to work well.

3.6 Risk estimation

A natural question is how large the number of sampled permutations K needs to be to obtain an accurate estimate of the Shapley values. In this section we provide a method to estimate the error in the Shapley attribution approximations provided by LS-SPA. The error estimates provide the user with an idea of the precision to which the attributions are accurate and can be used in a stopping criterion.

3.6.1 Error

We define the error in the estimate of the jth Shapley value to be

$$|\hat{S}_j - S_j|, \tag{11}$$

where $S \in \mathbf{R}^p$ is the vector of true Shapley values and $\hat{S} \in \mathbf{R}^p$ is the vector of approximate Shapley values as described in Sect. 2.3. We also define the overall error in the Shapley estimate to be

$$\|\hat{S} - S\|_2. \tag{12}$$

Other error metrics can be used, but (12) is a simple and default metric.

3.6.2 Risk estimation

Since the exact value of S is not known, (11) and (12) cannot be computed exactly. But, we can efficiently estimate the errors using the central limit theorem.

If a permutation π is sampled from the uniform distribution on $\mathcal{P}$, then the expected value of $L(\pi)$ is S. Let Σ denote the covariance of $L(\pi)$. The central limit theorem guarantees that $\sqrt{K}(\hat{S} - S)$ converges in distribution to $\mathcal{N}(0, \Sigma)$ as $K \to \infty$. We can thus estimate the qth quantile values of (11) and (12) over the distribution of $\hat{S}$ for K samples via Monte Carlo. We take $\hat{\Sigma}$ to be the unbiased sample covariance of $\{L(\pi)\}_{\pi\in\Pi}$, and sample D vectors $\Delta^{(1)}, \ldots, \Delta^{(D)}$

from $\mathcal{N}(0, \frac{1}{K}\hat{\Sigma})$. We then report the estimated error for feature j as

$$\hat{\rho}_j = \textbf{quantile}(\{|\Delta_j^{(i)}|\}_{i=1}^D; q)$$

and the estimated overall error as

$$\hat{\sigma} = \textbf{quantile}(\{\|\Delta^{(i)}\|_2\}_{i=1}^D; q),$$

where $\textbf{quantile}(\cdot; q)$ denotes the qth quantile. A higher value of q provides a more conservative error estimate.

3.6.3 Batching

To use the risk estimate in a stopping criterion, we can compute $\hat{S}$ in batches. After each batch, we recompute $\hat{\sigma}$ and terminate early if it is below a fixed tolerance $\epsilon > 0$. More precisely, we set a batch size B, a maximum number of batches K/B, and a risk tolerance $\epsilon > 0$.

For any subset Π of permutations, define the sample mean

$$\hat{S}(\Pi) = \frac{1}{|\Pi|}\sum_{\pi\in\Pi} L(\pi) \tag{13}$$

and the biased sample covariance

$$\hat{\Sigma}_b(\Pi) = \frac{1}{|\Pi|}\sum_{\pi\in\Pi}(L(\pi) - \hat{S}(\Pi))(L(\pi) - \hat{S}(\Pi))^T. \tag{14}$$

Instead of computing Π, $\hat{S}$, and the risk estimate all at once, we compute them iteratively via batches $\Pi^{(1)}, \ldots, \Pi^{(K/B)}$, each of size B. Initialize the estimated Shapley values $\hat{S}^{(0)} = 0$ and the estimated biased sample covariance $\hat{\Sigma}_b^{(0)} = 0$. In iteration j, we can compute $\hat{S}^{(j)}$ using the update rule

$$\hat{S}^{(j)} = \frac{j-1}{j}\hat{S}^{(j-1)} + \frac{1}{j}\hat{S}(\Pi^{(j)}), \tag{15}$$

which holds since $\Pi^{(1)}, \ldots, \Pi^{(K/B)}$ are equally sized. We can also compute $\hat{\Sigma}_b^{(j)}$ using the update rule provided in Schubert and Gertz (2018),

$$\hat{\Sigma}_b^{(j)} = \frac{j-1}{j}\hat{\Sigma}_b^{(j-1)} + \frac{1}{j}\hat{\Sigma}_b(\Pi^{(j)}) + \frac{j-1}{j^2}D^{(j)}D^{(j)T}, \tag{16}$$

where $D^{(j)} = \hat{S}^{(j-1)} - \hat{S}(\Pi^{(j)})$. The unbiased sample covariance $\hat{\Sigma}^{(j)}$ is $\frac{jB}{jB-1}\hat{\Sigma}_b^{(j)}$, which we can use to generate our risk estimates.

Note that batching in this manner can result in terminating early when $\hat{S}$ is computed on a number of permutations that is not a power of 2. When using Sobol' sequences, this can destroy the balance properties expected of QMC, but in practice, we have found this does not matter.

The central limit theorem is based on random samples, which is not the case for QMC methods. As a result, risk estimates when $\hat{S}$ is computed via a QMC method to sample permutations do not come with the theoretical guarantees that random samples have. We have observed empirically that estimates using QMC are still good estimates of the actual errors.

3.7 Sample augmentation

Monte Carlo and QMC sampling techniques can be augmented to potentially further reduce estimate variance. A simple way to do this is via antithetical sampling, in which for each permutation π sampled, the permutation $\gamma\pi$ is also included, where γ is the permutation that reverses the sequence $1, \ldots, p$. The permutation $\gamma\pi$ corresponds to the feature chain

$$\emptyset \subset \{\pi_k\} \subset \{\pi_k, \pi_{k-1}\} \subset \cdots \subset \{\pi_k, \pi_{k-1}, \ldots, \pi_1\}.$$

Note that if antithetical sampling is used, the sample mean (13) should be adjusted as

$$\hat{S}(\Pi) = \frac{1}{|\Pi|}\sum_{\pi\in\Pi}\tilde{L}(\pi),$$

and the sample covariance (14) should be adjusted to

$$\hat{\Sigma}_b(\Pi) = \frac{1}{|\Pi|}\sum_{\pi\in\Pi}(\tilde{L}(\pi) - \hat{S}(\Pi))(\tilde{L}(\pi) - \hat{S}(\Pi)),$$

where $\tilde{L}(\pi) = (L(\pi) + L(\gamma\pi))/2$. Empirically, we find that for well-conditioned data, antithetical sampling works very well when combined with Monte Carlo or QMC sampling and often converges more than twice as quickly. However, for data with poorly conditioned empirical covariance matrices, antithetical sampling gains little additional performance.

A more sophisticated technique is the ergodic sampling technique described in Illés and Kerényi (2022), which increases the number of permutations by shuffling each sampled permutation in a way that greedily minimizes the covariances of the lift vectors. However, this technique applied here introduces an $O(p^5)$ cost, so is not suitable for large p.

3.8 Algorithm summary

The LS-SPA algorithm is summarized in algorithm 1. We note that in line 2, the Cholesky reduction described in Sect. 4.3 may be used instead of the QR reduction described in Sect. 3.2.

Result: Return $\hat{S}$, $\{\hat{\rho}_i\}_{i=1}^p$, and $\hat{\sigma}$
Given: Training data $X \in \mathbf{R}^{N\times p}$, training labels $y \in \mathbf{R}^N$, test data $X^{\text{tst}} \in \mathbf{R}^{M\times p}$, test labels $y^{\text{tst}} \in \mathbf{R}^M$, maximum number of sampled permutations $K \in \mathbf{Z}_{++}$, batch size $B \in \mathbf{Z}_{++}$, risk tolerance $\epsilon \in \mathbf{R}_{++}$, error quantile $q \in (0, 1)$;
Reduce X, y, X^{tst}, y^{tst} as described in Sect. 3.2;
Generate K permutations $\pi^{(1)}, \ldots, \pi^{(K)}$ as described in Sect. 2.3 or §3.5;
for $j = 1, \ldots, K/B$ **do**
  for $k = (j-1)B + 1, \ldots, jB$ **do**
    Compute lifts $L(\pi^{(k)})$ as described in Sect. 3.2, and if antithetical sampling is used, also compute $L(\gamma\pi^{(k)})$;
  end
  Compute approximate attributions $\hat{S}^{(j)}$ and estimated overall errors $\hat{\sigma}$ as described in Sect. 3.6;
  if error estimate below tolerance, $\hat{\sigma} < \epsilon$ **then**
    Compute estimated feature errors $\{\hat{\rho}_i\}_{i=1}^p$ as described in Sect. 3.6;
    return $\hat{S} = \hat{S}^{(j)}$, $\{\hat{\rho}_i\}_{i=1}^p$, $\hat{\sigma}$
  end
end
Print tolerance not reached warning;
Compute $\hat{S}^{(j)}$ and $\hat{\sigma}$ as described in Sect. 3.6;
Compute $\{\hat{\rho}_i\}_{i=1}^p$ as described in Sect. 3.6;
return $\hat{S} = \hat{S}^{(K/B)}$, $\{\hat{\rho}_i\}_{i=1}^p$, $\hat{\sigma}$;

Algorithm 1: LEAST- SQUARES SHAPLEY ATTRIBUTION (LS- SPA)

3.9 Implementation

We have developed two Python implementations of algorithm 1. The computational results we present in Sect. 5 are derived from a JAX-based (Bradbury et al. 2023) implementation of algorithm 1 and some of the extensions discussed in Sect. 4. The JAX implementation, along with our numerical experiments, is available at

https://github.com/cvxgrp/ls-spa-benchmark.

We also provide a more user-friendly, NumPy-based (Harris et al. 2020) library implementing algorithm 1 at

https://github.com/cvxgrp/ls-spa.

JAX allows LS-SPA to utilize GPU(s), while the NumPy implementation of LS-SPA runs on CPUs only. In addition, the JAX implementation employs some additional parallelization to execute LS-SPA more efficiently, although as a consequence, the JAX implementation is harder to read and harder to use. In addition, JAX is harder to install and configure, especially in order to use GPU(s). For this reason, we provide the NumPy implementation, which lacks some of the features (notably QMC) and performance of the JAX implementation, but is in turn much easier to install, read, and use.

4 Extensions and variations

In this section, we describe some extensions to the basic problem and method described above.

4.1 Cross validation metric

In the discussion above we used simple out-of-sample validation, but we can also use other more sophisticated validation methods, such as M-fold cross validation (Efron and Tibshirani 1993, Ch. 17). Here the original data are split into M different 'folds'. For $m = 1, \ldots, M$ we fit a model using as training data all folds except m and validate it on fold m. We use the average validation mean-square error to obtain the R^2 score. The methods above apply immediately to this situation.

4.2 Ridge regularization

In ridge regression, we choose the parameter θ by solving the ℓ_2-regularized least-squares problem

$$\text{minimize } \frac{1}{N}\|X\theta - y\|_2^2 + \lambda\|\theta\|_2^2, \tag{17}$$

where $\theta \in \mathbf{R}^p$ is the optimization variable, $X \in \mathbf{R}^{N\times p}$ and $y \in \mathbf{R}^N$ are data, and λ is a positive regularization hyperparameter. Observe that (17) can be reformulated as

$$\text{minimize } \|\tilde{X}\theta - \tilde{y}\|_2^2, \tag{18}$$

where $\tilde{X}$ and $\tilde{y}$ are the stacked data

$$\tilde{X} = \begin{bmatrix} X/\sqrt{N} \\ \sqrt{\lambda}I \end{bmatrix}, \qquad \tilde{y} = \begin{bmatrix} y/\sqrt{N} \\ 0 \end{bmatrix}.$$

This reformulation transforms the regularized problem (17) into a least-squares problem in the form of (1). As such, we can now perform LS-SPA on the regularized problem.

To choose the value of the hyper-parameter λ, we consider a set of candidate values $\lambda_1, \ldots, \lambda_L$. We solve the regularized least-squares regression problem for each one and evaluate the resulting parameter λ using out-of-sample or cross-validation. We then choose λ as the one among our choices that achieves the lower mean-square test error. We use this value to compute the R^2 metric.

4.3 Very large data

If X is too large to fit into memory such that performing the initial QR factorization cannot be done, one alternative is to compute the Cholesky factorization of the covariance matrix of $[X\ y]$, *i.e.*, the matrix

$$\hat{\Sigma} = \begin{bmatrix} X^T \\ y^T \end{bmatrix} \begin{bmatrix} X & y \end{bmatrix} = \begin{bmatrix} X^T X & X^T y \\ y^T X & y^T y \end{bmatrix}.$$

The covariance matrix $\hat{\Sigma}$ is $p \times p$ and can be computed via block matrix multiplication by blocking $[X\ y]$ vertically, making it possible to distribute the computation across multiple devices or compute iteratively on one device. The upper-triangular factor $\tilde{R}$ in the Cholesky factorization $\tilde{R}^T \tilde{R} = \hat{\Sigma}$ can then be blocked as

$$\tilde{R} = \begin{bmatrix} R & Q^T y \\ 0 & \|y - Q(Q^T y)\|_2 \end{bmatrix}$$

where $QR = X$ is the QR factorization of X. We can thus extract R, $Q^T y$, and $\|y - Q(Q^T y)\|_2$ from $\tilde{R}$ to compute the reduction (7) for use in LS-SPA. This alternative approach costs $O(Np^2)$ flops for the computation of $\hat{\Sigma}$ and $O(p^3)$ flops for the computation of $\tilde{R}$, giving a total cost of $O(Np^2)$, the same as the QR method. However, Cholesky factorization is less stable than QR and can fail for poorly conditioned $\hat{\Sigma}$.

4.4 Non-quadratic regularizers

We consider the case where the quadratic loss is paired with a non-quadratic but convex regularizer. This means we choose the model parameter θ by solving

$$\text{minimize } \|X\theta - y\|_2^2 + \lambda r(\theta), \tag{19}$$

with variable $\theta \in \mathbf{R}^p$, data $X \in \mathbf{R}^{N \times p}$ and $y \in \mathbf{R}^N$, and convex but non-quadratic regularizer $r : \mathbf{R}^p \to \mathbf{R} \cup \{\infty\}$. Here λ is the regularization hyper-parameter. Simple examples include the nonnegative indicator function, so the problem above is a non-negative least-squares problem. Another example is $r(\theta) = \|\theta\|_1$, which gives the lasso problem (Hastie et al. 2009).

While our formula for θ given in Sect. 3.2 no longer holds, we can still reduce the complexity of the computation with the initial reduction. Thus when we find θ we solve a smaller convex optimization problem with a square data matrix.

5 Numerical experiments

5.1 Experiment descriptions

We describe two numerical experiments, one medium size and one large, that demonstrate the relationship between the runtime of the LS-SPA and the accuracy of the approximated Shapley attribution. The code for the experiments can be found in

https://github.com/cvxgrp/ls-spa-benchmark.

5.1.1 Medium size experiment

The medium-size experiment uses a single randomly generated data set with $p = 100$ features and $N = M = 10^5$ data points for the train and test data sets. Details of data generation are given in Sect. 5.2. The medium-size experiment is meant to show how the overall error in the estimate of the Shapley attributions evolves with an increasing number of sampled feature chains. We run LS-SPA once with of each of three methods to sample feature chains (MC, permutohedron QMC, and argsort QMC) in the medium size experiment, with and without antithetical sampling, for a total of $K = 2^{13}$ sampled permutations. For the QMC methods, we use scrambled Sobol' sequences with SciPy's default scrambling strategy, which is a (left) linear matrix scramble followed by a digital random shift (Virtanen et al. 2020; Matoušek 1998). We track the estimated overall error and the true overall error as more permutations are sampled during the runtime of LS-SPA. We compare the true overall errors achieved by each sampling method, and we compare the error estimate to the true error for MC and argsort QMC. For the purpose of computing the true overall error, we compute the "ground-truth" Shapley attributions by running LS-SPA with Monte Carlo and antithetical sampling for 2^{28} total permutations. The quantile we use for risk estimation is $q = 0.95$.

5.1.2 Large experiment

The large experiment uses a single randomly generated data set with $p = 1000$ features and $N = M = 10^6$ data points for the train and test data sets. Details of data generation are given in Sect. 5.2. The large experiment is a timing test meant to demonstrate that LS-SPA scales to large problems. The large experiment uses a single run of LS-SPA with Monte Carlo and antithetical sampling only and is run until the error estimate falls below a tolerance $\epsilon = 10^{-3}$. The quantile we use for risk estimation is $q = 0.95$.

5.1.3 Computation platforms

The medium-size experiment, except for computation of the ground-truth Shapley attributions, was done on a 16-thread Intel Core i7-10875 H CPU at 2.30 GHz with 64 GB RAM. The large experiment and computation of ground truth for the medium experiment were done with two Intel Xeon E5-2640 v4 CPUs, each with 20 threads, and four NVIDIA GTX TITAN X GPUs, each of which has 12 GB RAM. Note that for the large experiment and computation of ground truth for the medium experiment, all numerical computations were done on GPU.

5.2 Data generation

For both experiments, we solved instances of (1) on randomly generated train and test data, $(X^{\text{trn}}, y^{\text{trn}})$ and $(X^{\text{tst}}, y^{\text{tst}})$, respectively. To generate the data, we first randomly generate a feature covariance matrix $\Sigma = FF^T + I$, where $F \in \mathbf{R}^{p\times(p/20)}$ is generated by sampling its entries independently from a $\mathcal{N}(0, 1)$ distribution. We then let C be the correlation matrix of Σ.

Next, the true vector of feature coefficients θ was generated by randomly selecting $\lfloor (p+1)/10 \rfloor$ entries to be 2 and the remaining entries to be 0.

Finally, we generate $X^{\text{trn}} \in \mathbf{R}^{N\times p}$ and $X^{\text{tst}} \in \mathbf{R}^{M\times p}$, consisting, respectively, of N and M observations generated independently at random from a $\mathcal{N}(0, C)$ distribution. We then generate noise vectors $\omega^{\text{trn}}, \omega^{\text{tst}} \in \mathbf{R}^p$ independently from a $\mathcal{N}(0, (3p^2/2)I)$ distribution and construct $y^{\text{trn}} = X^{\text{trn}}\theta + \omega^{\text{trn}}$ and $y^{\text{tst}} = X^{\text{tst}}\theta + \omega^{\text{tst}}$. Finally, we include an intercept in our linear model by centering the columns of X^{trn} and X^{tst} by subtracting the respective column means of X^{trn}, and also centering y^{trn} and y^{tst} by subtracting the mean of y^{trn}, as discussed in Sect. 2.1. The features generated had high correlation, which we found empirically was adversarial for LS-SPA.

5.3 Results

5.3.1 Medium size experiment

We used each of MC, permutohedron QMC, and argsort QMC, with and without antithetical sampling, to sample $K = 2^{13}$ total feature chains, done in 2^5 batches of size 2^8 to illustrate the progress of LS-SPA as more permutations are sampled. LS-SPA took an average of 9 min 12 s to compute 2^{13} lift vectors, which is 67.4 milliseconds per lift vector. In comparison, a naïve implementation that does not take advantage of any reductions described in LS-SPA took 4 min 26 s to compute 2^3 lift vectors, which is 33.3 s per lift vector. The errors for each method sampling method as a function of the number of feature chains completed are shown in Figs. 2 and 3. Note that the condition number of C was 316.0.

In Fig. 4, we also plot the true overall error against the error estimate, which was computed using the risk estimation procedure described in Sect. 3.6, at each step of the algorithm using Monte Carlo with antithetical sampling to sample feature chains. We plot the same things in Fig. 5 using argsort QMC without antithetical sampling to sample feature chains.

5.3.2 Large experiment

We use Monte Carlo with antithetical sampling to sample feature chains and run LS-SPA until the estimated error $\hat{\sigma}$ is below the tolerance level $\epsilon = 10^{-3}$. We use a quantile value

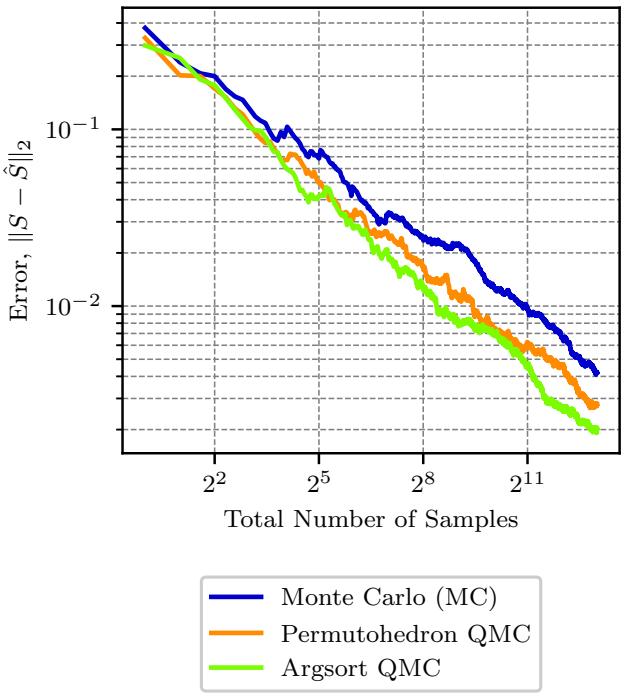

Fig. 2 Overall error versus number of samples on the medium-size dataset using MC (blue), permutohedron QMC (orange), and argsort QMC (green) without antithetical sampling to sample feature chains. (Color figure online)

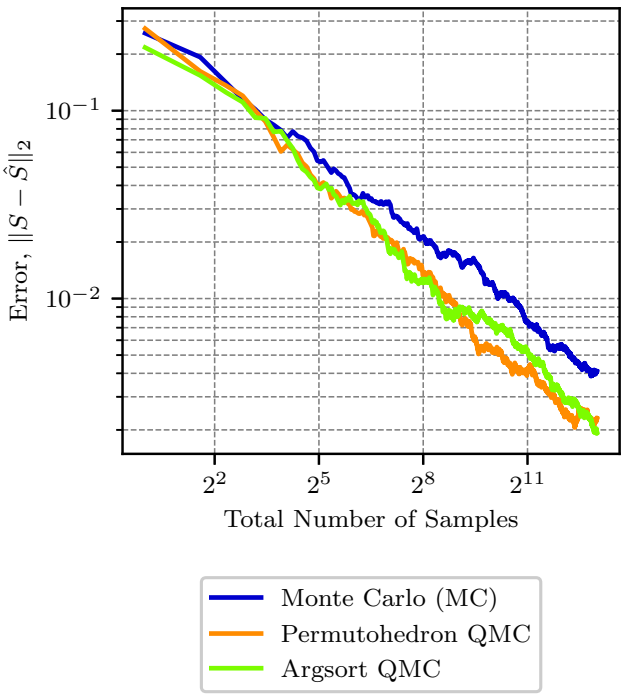

Fig. 3 Overall error versus number of samples on the medium-size dataset using MC (blue), permutohedron QMC (orange), and argsort QMC (green) with antithetical sampling to sample feature chains. (Color figure online)

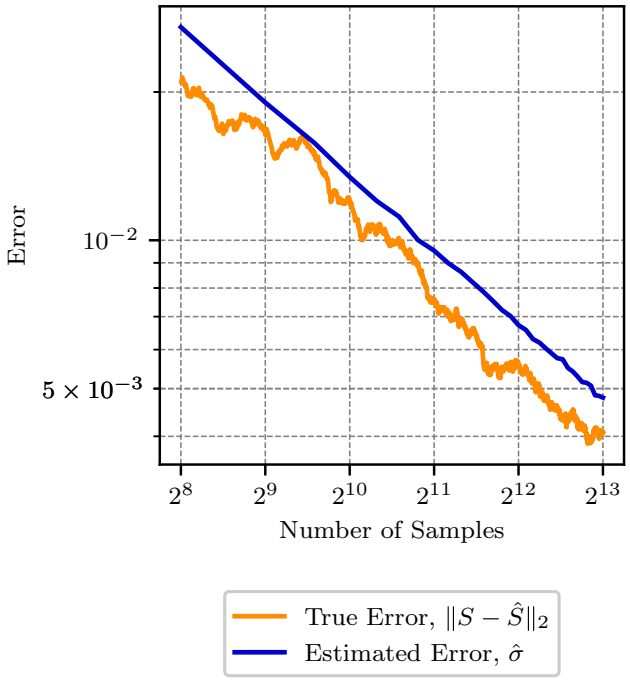

Fig. 4 True error (blue) and estimated error (orange) while running LS-SPA using Monte Carlo with antithetical sampling to sample feature chains. (Color figure online)

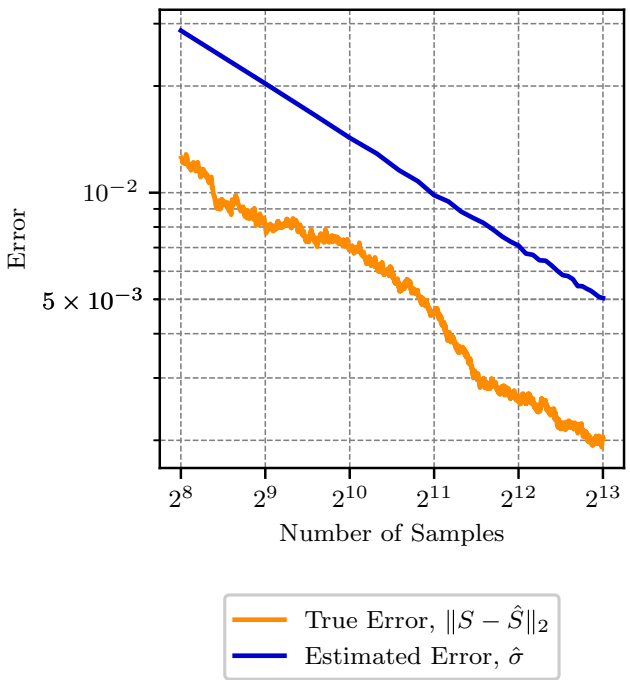

Fig. 5 True error (blue) and estimated error (orange) while running LS-SPA using argsort QMC without antithetical sampling to sample feature chains. (Color figure online)

of $q = 0.95$. Since the data were too large to fit on one GPU, we use the Cholesky reduction presented in Sect. 4.3.

The algorithm took 27.2 s to complete the initial reduction. LS-SPA ran for 230.4 s to reach an error estimate of 9.9×10^{-4}, computing a total of 29,696 lift vectors, done in 29 batches of 2^8 permutations on each of the four GPUs. This gives an average of 7.8×10^{-3} s per lift vector. Note that the correlation matrix C used to generate the data has condition number 2.4×10^3.

5.4 Discussion

For moderately sized p, e.g., on the order of 100, the NumPy implementation of LS-SPA linked in Sect. 3.9 fairly quickly converges to an estimate of the Shapley attributions with error 10^{-3}. This is true even when using Monte Carlo sampling, which tends to underperform compared to quasi-Monte Carlo sampling techniques. For p of this size, LS-SPA achieves a 500× speedup compared to a naïve estimation procedure.

For larger p on the order of 1000 or more, the NumPy implementation can still be used, but a more specialized implementation should be used for maximum performance. An example JAX implementation is available in the benchmark repo linked in Sect. 5.1.

Acknowledgements We thank Ron Kahn for suggesting the topic, Kunal Menda for recommending the use of quasi-Monte Carlo, Trevor Hastie and Emmanuel Candès for suggesting the risk estimation method, and Art Owen and Thomas Schmelzer for helpful feedback on a draft. We are indebted to anonymous reviewers for pointing us to literature we had missed and in addition suggesting methods such as antithetical sampling.

Author Contributions All authors contributed to the algorithm design, analysis, testing, and writing of the manuscript. All authors read and approved the final manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

References

Algaba, E., Fragnelli, V., Sánchez-Soriano, J.: Handbook of the Shapley Value. Chapman & Hall/CRC, Boca Raton (2019). https://doi.org/10.1201/9781351241410

Boyd, S., Vandenberghe, L.: Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares. Cambridge University Press, Cambridge (2018). https://doi.org/10.1017/9781108583664

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q.: JAX: Composable transformations of Python+NumPy programs (2023). http://github.com/google/jax

Budescu, D.: Dominance analysis: A new approach to the problem of relative importance of predictors in multiple regression. Psychol.

Bull. **114**(3), 542–551 (1993). https://doi.org/10.1037/0033-2909.114.3.542

Businger, P., Golub, G.: Linear least squares solutions by householder transformations. Numer. Math. **7**(3), 269–276 (1965). https://doi.org/10.1007/bf01436084

Campen, T., Hamers, H., Husslage, B., Lindelauf, R.: A new approximation method for the Shapley value applied to the WTC 9/11 terrorist attack. Soc. Netw. Anal. Min. (2017). https://doi.org/10.1007/s13278-017-0480-z

Castro, J., Gómez, D., Tejada, J.: Polynomial calculation of the Shapley value based on sampling. Comput. Oper. Res. **36**(5), 1726–1730 (2009). https://doi.org/10.1016/j.cor.2008.04.004

Castro, J., Gómez, D., Molina, E., Tejada, J.: Improving polynomial estimation of the Shapley value by stratified random sampling with optimum allocation. Comput. Oper. Res. **82**, 180–188 (2017). https://doi.org/10.1016/j.cor.2017.01.019

Chalkiadakis, G., Elkind, E., Wooldridge, M.: Computational Aspects of Cooperative Game Theory. Springer, Cham (2012). https://doi.org/10.1007/978-3-031-01558-8

Chen, H., Covert, I., Lundberg, S., Lee, S.-I.: Algorithms to estimate Shapley value feature attributions. Nat. Mach. Intell. **5**(6), 590–601 (2023). https://doi.org/10.1038/s42256-023-00657-x

Chevan, A., Sutherland, M.: Hierarchical partitioning. Am. Stat. **45**(2), 90–96 (1991). https://doi.org/10.2307/2684366

Curiel, I., Pederzoli, G., Tijs, S.: Sequencing games. Eur. J. Oper. Res. **40**(3), 344–351 (1989). https://doi.org/10.1016/0377-2217(89)90427-x

Deng, X., Papadimitriou, C.: On the complexity of cooperative solution concepts. Math. Oper. Res. **19**(2), 257–266 (1994). https://doi.org/10.1287/moor.19.2.257

Dubey, P., Neyman, A., Weber, R.: Value theory without efficiency. Math. Oper. Res. **6**(1), 122–128 (1981)

Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. Monographs on Statistics and Applied Probability, vol. 57. Chapman & Hall/CRC, Boca Raton (1993)

Faigle, U., Kern, W.: The Shapley value for cooperative games under precedence constraints. Int. J. Game Theory **21**(3), 249–266 (1992). https://doi.org/10.1007/bf01258278

Fatima, S., Wooldridge, M., Jennings, N.: A linear approximation method for the Shapley value. Artif. Intell. **172**(14), 1673–1699 (2008). https://doi.org/10.1016/j.artint.2008.05.003

Fernández, J., Algaba, E., Bilbao, J., Jiménez, A., Jiménez, N., López, J.: Generating functions for computing the Myerson value. Ann. Oper. Res. **109**(1/4), 143–158 (2002). https://doi.org/10.1023/a:1016348001805

Frisch, R., Waugh, F.: Partial time regressions as compared with individual trends. Econometrica **1**, 387 (1933)

Fryer, D., Strümke, I., Nguyen, H.: Shapley values for feature selection: the good, the bad, and the axioms. IEEE Access **9**, 144352–144360 (2021). https://doi.org/10.1109/ACCESS.2021.3119110

Ghorbani, A., Zou, J.: Data Shapley: Equitable valuation of data for machine learning. In: Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 2242–2251. PMLR, Long Beach, CA, USA (2019). https://proceedings.mlr.press/v97/ghorbani19c.html

Golub, G.: Numerical methods for solving linear least squares problems. Numer. Math. **7**(3), 206–216 (1965). https://doi.org/10.1007/bf01436075

Granot, D., Kuipers, J., Chopra, S.: Cost allocation for a tree network with heterogeneous customers. Math. Oper. Res. **27**(4), 647–661 (2002). https://doi.org/10.1287/moor.27.4.647.307

Grömping, U.: Relative importance for linear regression in R: The package relaimpo. J. Stat. Softw. (2006). https://doi.org/10.18637/jss.v017.i01

Grömping, U.: Variable importance in regression models. WIREs Comput. Stat. **7**(2), 137–152 (2015). https://doi.org/10.1002/wics.1346

Harris, C., Pymar, R., Rowat, C.: Joint Shapley values: a measure of joint feature importance. In: International Conference on Learning Representations (2022). https://doi.org/10.48550/arXiv.2107.11357

Harris, C.R., Millman, K.J., Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M.H., Brett, M., Haldane, A., Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2

Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer, New York City (2009). https://doi.org/10.1007/978-0-387-84858-7

Huettner, F., Sunder, M.: Axiomatic arguments for decomposing goodness of fit according to Shapley and Owen values. Electron. J. Stat. **6**, 1239–1250 (2012). https://doi.org/10.1214/12-EJS710

Ieong, S., Shoham, Y.: Marginal contribution nets. In: Proceedings of the 6th ACM Conference on Electronic Commerce. ACM, Vancouver, British Columbia, Canada (2005). https://doi.org/10.1145/1064009.1064030

Illés, F., Kerényi, P.: Estimation of the Shapley value by ergodic sampling (2022)

Kóczy, L.: A recursive core for partition function form games. Theor. Decis. **63**(1), 41–51 (2007). https://doi.org/10.1007/s11238-007-9030-x

Kruskal, W.: Relative importance by averaging over orderings. Am. Stat. **41**, 6–10 (1987). https://doi.org/10.1080/00031305.1987.10475432

Kumar, I.E., Venkatasubramanian, S., Scheidegger, C., Friedler, S.: Problems with Shapley-value-based explanations as feature importance measures. In: Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 5491–5500. PMLR, online (2020). https://proceedings.mlr.press/v119/kumar20e.html

Leech, D.: Computing power indices for large voting games. Manage. Sci. **49**(6), 831–837 (2003). https://doi.org/10.1287/mnsc.49.6.831.16024

Lindeman, R., Merenda, P., Gold, R.: Introduction to Bivariate and Multivariate Analysis. Scott Foresman, Northbrook, IL, USA (1980). https://books.google.com/books?id=-hfvAAAAMAAJ

Lipovetsky, S., Conklin, M.: Analysis of regression in game theory approach. Appl. Stoch. Models Bus. Ind. **17**(4), 319–330 (2001). https://doi.org/10.1002/asmb.446

Littlechild, S., Owen, G.: A simple expression for the Shapley value in a special case. Manage. Sci. **20**(3), 370–372 (1973). https://doi.org/10.1287/mnsc.20.3.370

Lovell, M.: Seasonal adjustment of economic time series and multiple regression analysis. J. Am. Stat. Assoc. **58**(304), 993–1010 (1963). https://doi.org/10.1080/01621459.1963.10480682

Lundberg, S., Lee, S.-I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems 30, pp. 4765–4774. Curran Associates, Inc., Long Beach, CA, USA (2017)

Maleki, S., Tran-Thanh, L., Hines, G., Rahwan, T., Rogers, A.: Bounding the Estimation Error of Sampling-based Shapley Value Approximation. arXiv:1306.4265 (2014)

Mann, I., Shapley, L.: Values of Large Games, IV: Evaluating the Electoral College by Montecarlo Techniques. RAND Corporation, Santa Monica, CA (1960)

Matoušek, J.: On the L_2-discrepancy for anchored boxes. J. Complex. **14**, 527–556 (1998). https://doi.org/10.1006/jcom.1998.0489

Michalak, T., Rahwan, T., Szczepanski, P., Skibski, O., Narayanam, R., Jennings, N., Wooldridge, M.: Computational analysis of connectivity games with applications to the investigation of terrorist

networks. In: International Joint Conference on Artificial Intelligence (2013)
Michalak, T., Aadithya, K., Szczepanski, P., Ravindran, B., Jennings, N.: Efficient computation of the Shapley value for game-theoretic network centrality. J. Artif. Intell. Res. **46**, 607–650 (2013). https://doi.org/10.1613/jair.3806
Mishra, S.: Shapley value regression and the resolution of multicollinearity. SSRN Electron. J. (2016). https://doi.org/10.2139/ssrn.2797224
Mitchell, R., Cooper, J., Frank, E., Holmes, G.: Sampling permutations for Shapley value estimation. J. Mach. Learn. Res. **23**(43), 1–46 (2022)
Moehle, N., Boyd, S., Ang, A.: Portfolio performance attribution via Shapley value. J. Invest. Manage. **20**(3), 33–52 (2022)
Molnar, C.: Interpretable Machine Learning (2022). https://christophm.github.io/interpretable-ml-book
Monderer, D., Samet, D.: Variations on the Shapley value. In: Handbook of Game Theory with Economic Applications Volume 3. Handbook of Game Theory with Economic Applications, vol. 3, pp. 2055–2076. Elsevier, Amsterdam, Netherlands (2002). Chap. 54. https://doi.org/10.1016/S1574-0005(02)03017-5
Moretti, S., Patrone, F.: Transversality of the Shapley value. TOP **16**(1), 1–41 (2008). https://doi.org/10.1007/s11750-008-0044-5
Owen, A.B.: Practical Quasi-Monte Carlo Integration. https://artowen.su.domains/mc/practicalqmc.pdf, (2023)
Owen, G.: Values of games with a priori unions. In: Mathematical Economics and Game Theory, pp. 76–88. Springer, Berlin, Heidelberg (1977)
Owen, G.: Multilinear extensions of games. Manage. Sci. **18**(5), 64–79 (1972)
Owen, A.B.: Scrambling sobol' and niederreiter-xing points. J. Complex. **14**, 466–489 (1998). https://doi.org/10.1006/jcom.1998.0487
Owen, A., Prieur, C.: On Shapley value for measuring importance of dependent inputs. SIAM/ASA J. Uncertain. Quantif. **5**(1), 986–1002 (2017). https://doi.org/10.1137/16M1097717
Powers, M.: Using Aumann-Shapley values to allocate insurance risk. N. Am. Actuar. J. **11**(3), 113–127 (2007). https://doi.org/10.1080/10920277.2007.10597470
Schubert, E., Gertz, M.: Numerically stable parallel computation of (co-)variance. In: Proceedings of the 30th International Conference on Scientific and Statistical Database Management. SSDBM '18. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3221269.3223036
Shapley, L.: A value for N-person games. In: Contributions to the Theory of Games (AM-28), Vol. II, pp. 307–318. Princeton University Press, Princeton, NJ, USA (1952). https://doi.org/10.1515/9781400881970-018
Stufken, J.: Letters to the editor: on hierarchical partitioning. Am. Stat. **46**(1), 70–77 (1992)
Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., Mulbregt, P., Vijaykumar, A., Bardelli, A.P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C.N., Fulton, C., Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D.A., Hagen, D.R., Pasechnik, D.V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G.A., Ingold, G.-L., Allen, G.E., Lee, G.R., Audren, H., Probst, I., Dietrich, J.P., Silterra, J., Webber, J.T., Slavič, J., Nothman, J., Buchner, J., Kulick, J., Schönberger, J.L., Miranda Cardoso, J.V., Reimer, J., Harrington, J., Rodríguez, J.L.C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N.J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P.A., Lee, P., McGibbon, R.T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T.J., Robitaille, T.P., Spura, T., Jones, T.R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y.O., Vázquez-Baeza, Y.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nat. Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
Wang, J., Zhang, Y., Kim, T.-K., Gu, Y.: Shapley Q-value: a local reward approach to solve global reward games. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05, pp. 7285–7292 (2020). https://doi.org/10.1609/aaai.v34i05.6220
Williamson, B., Feng, J.: Efficient nonparametric statistical inference on population feature importance using shapley values. In: Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 10282–10291. PMLR, online (2020). https://proceedings.mlr.press/v119/williamson20a.html
Zhang, H., Singh, H., Ghassemi, M., Joshi, S.: Why did the model fail?: Attributing model performance changes to distribution shifts. In: Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 41550–41578. PMLR, Honolulu, HI, USA (2023). https://proceedings.mlr.press/v202/zhang23ai.html
Zhao, K., Mahboobi, S.H., Bagheri, S.: Shapley Value Methods for Attribution Modeling in Online Advertising (2018) arXiv:1804.05327
Zlotkin, G., Rosenschein, J.: Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. In: Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence. AAAI'94, pp. 432–437. AAAI Press, Seattle, WA, USA (1994)