SHAP@k: Efficient and Probably Approximately Correct (PAC) Identification of Top-k Features

Sanjay Kariyappa, Leonidas Tsepenekas, Freddy Lécué, Daniele Magazzeni

JPMorganChase AI Research

{sanjay.kariyappa, leonidas.tsepenekas, freddy.lecue, daniele.magazzeni}@jpmorgan.com

Abstract

The SHAP framework provides a principled method to explain the predictions of a model by computing feature importance. Motivated by applications in finance, we introduce the *Top-k Identification Problem (TkIP)* (and its ordered variant TkIP-O), where the objective is to identify the subset (or ordered subset for TkIP-O) of k features corresponding to the highest SHAP values with PAC guarantees. While any sampling-based method that estimates SHAP values (such as KernelSHAP and SamplingSHAP) can be trivially adapted to solve TkIP, doing so is highly sample inefficient. Instead, we leverage the connection between SHAP values and multi-armed bandits (MAB) to show that both TkIP and TkIP-O can be reduced to variants of problems in MAB literature. This reduction allows us to use insights from the MAB literature to develop sample-efficient variants of KernelSHAP and SamplingSHAP. We propose *KernelSHAP@k* and *SamplingSHAP@k* for solving TkIP; along with *KernelSHAP-O* and *SamplingSHAP-O* to solve the ordering problem in TkIP-O. We perform extensive experiments using several credit-related datasets to show that our methods offer significant improvements of up to $40\times$ in sample efficiency and $39\times$ in runtime.

1 Introduction

The ability to explain the predictions of ML models is of critical importance in highly regulated industries, where laws provide *right to explanations* for people who are adversely impacted by algorithmic decision making. Specifically in finance, regulations like Fair Credit Reporting Act (Congress 1970) and Equal Credit Opportunity Act (Congress 1974) require a rejected loan/credit application (i.e. adverse action) to be explained to the borrower, by providing reasons for why the application was rejected (e.g., low credit score, high debt-to-income ratio, recent delinquencies, etc.). Owing to its principled formulation, the SHAP framework (Lundberg and Lee 2017) is the defacto choice for explaining model predictions in credit-risk assesment models (Bhatt et al. 2020). While exact computation of SHAP values is computationally intractable, sampling-based techniques like KernelSHAP (Lundberg and Lee 2017) and SamplingSHAP provide a practical alternative to compute approximate SHAP values. Additionally, recent works have developed methods to quantify the approximation error of such sampling-based techniques, by providing confidence intervals (CIs) for the estimated SHAP values (Covert and Lee 2020).

Motivated by applications in finance, we introduce two new problems with the following objectives:

1. *Top-k Identification Problem (TkIP):* Identify an unordered set of k features associated with the k highest SHAP values with PAC guarantees[1].
2. *Ordered Top-k Identification Problem (TkIP-O):* Identify an ordered set of k features associated with the k highest SHAP values with PAC guarantees.

TkIP and TkIP-O are motivated by an important real-world use-case of processing credit/loan applications, where the lender is required to provide a list of up to four specific principal reasons or features that contributed to the model's prediction (i.e. explanations) in the event of an adverse decision/action taken; this is standard practice by credit/loan issuers to comply with the Equal Credit Opportunity Act (Congress 1974). While there are no prior works that solve TkIP/TkIP-O specifically, there is a recent prior work (Kolpaczki, Bengs, and Hüllermeier 2021) that has looked at solving a variant of TkIP with a fixed sample budget. However, this work does not provide PAC guarantees for its solution and it does not solve the ordering problem introduced in TkIP-O.

Naive baseline. We show that existing sampling-based methods like Kernel/SamplingSHAP can be straightforwardly adapted to solve TkIP/TkIP-O. We design a naive stopping condition that provides PAC solutions by requiring the CI-widths for all features to fall below a threshold. While this provides valid solutions for TkIP/TkIP-O, we show that this naive baseline requires a large number of samples, emphasizing the need for sample-efficient methods.

Connection to MAB. We develop a sample-efficient solution for TkIP by observing that this problem can be reduced to a stricter variant of the *Explore-m* problem in multi-arm bandits (MAB), where given d arms with an unknown distribution of payoffs, the goal is to find a subset of m-best arms with the highest expected payoffs. Here, the features are analogous to arms and the SHAP value of each feature is analogous to the expected payoff of each arm.

Similarly, TkIP-O is related to the *ordering problem* in MAB, where given a set of d arms, the goal is to sort them

[1] Formal definitions of PAC guarantees are provided in Section 3.

according to their expected payoffs. One approach for solving TkIP-O is to sort all d features according to their SHAP values (by solving the ordering problem) and just returning the top-k features. However, this direct application of the ordering problem is sub-optimal as it unnecessarily orders all d features. We develop sample-efficient solutions by leveraging the insight that TkIP-O can be decomposed into two sub-problems:

P1. Identify an unordered set of k features with the k-highest SHAP values. (*TkIP*)

P2. Order these k features in decreasing order of their SHAP values. (*ordering problem*)

We formally prove (in Theorem 3) that the above decomposition provides a PAC solution and use it to develop sample-efficient methods to solve TkIP-O

Our proposal for solving TkIP. We adapt KernelSHAP and SamplingSHAP by leveraging the connection between TkIP and the *Explore-m* problem. We start by showing (in Theorem 2) that the LUCB (Kalyanakrishnan et al. 2012) algorithm (a popular solution for the Explore-m problem) satisfies the stricter requirement for identifying *top-k features* that is introduced by TkIP. This allows us to use LUCB's stopping condition and sampling scheme to modify kernel/SamplingSHAP to develop sample efficient variants–*KernelSHAP@k* and *SamplingSHAP@k*– for solving TkIP.

Our proposal for solving TkIP-O. We solve TkIP-O by first using Kernel/SamplingSHAP@k to solve TkIP (*P1*), and then ordering the top-k features according to their SHAP values. To solve the ordering problem (*P2*) efficiently, we leverage the LUCBRank (Katariya et al. 2018) algorithm by modifying the stopping condition and sampling schemes of Kernel/SamplingSHAP to develop KernelSHAP-O and SamplingSHAP-O.

To the best of our knowledge, we are the first to provide sample-efficient solutions for TkIP and TkIP-O with PAC guarantees. We evaluate our proposed methods using a suite of credit-related datasets and quantify the relative performance between our different proposed methods, and their performance relative to the naive baseline. We find that SamplingSHAP@k/SamplingSHAP-O offers the best performance for TkIP/TkIP-O, with improvements of up to $40\times$ in sample efficiency and $39\times$ in runtime compared to the naive baseline. The rest of this paper is structured as follows:

- In Section 2, we provide background on sampling-based methods that can be used to estimate SHAP values and related work on variance reduction, uncertainty estimation and efficient estimation of SHAP values.
- In Section 3, we formally define the *Top-k Identification problem (TkIP)* and *Ordered Top-k Identification problem (TkIP-O)*, and develop a *naive stopping condition* that can be used with Kernel/Sampling SHAP to solve TkIP/TkIP-O. We illustrate the sample inefficiencies of this method using an example, motivating the need for better methods.
- In Section 4, we develop *Kernel/SamplingSHAP@k* for solving TkIP and *Kernel/SamplingSHAP-O* to solve the ordering problem (required for solving TkIP-O) by leveraging algorithms from the MAB literature.
- In Section 5, we evaluate our proposed methods on a suite of credit-related datasets to demonstrate the improvements in sample costs for our proposed methods and measure sensitivity to various parameters.
- We discuss limitations in Section 6 and conclude in Section 7. Proofs, algorithms and sensitivity studies can be found in the Appendix.

2 Background and Related Work

The goal of our work is to modify existing sampling-based SHAP estimation techniques to efficiently solve TkIP and TkIP-O (with PAC guarantees). In this section, we provide background on the SHAP framework and discuss existing sampling-based techniques (SamplingSHAP and KernelSHAP) that estimate SHAP values. Additionally, we discuss related works that extend these methods by reducing the variance of the estimates and quantify uncertainty in the form of confidence intervals. We also briefly discuss other prior works on efficiently estimating SHAP values, without guarantees on accuracy. Note that, in the context of TkIP/TkIP-O, all prior works suffer from at least one of the following limitations: 1. They are not designed to solve TkIP/TkIP-O efficiently 2. They do not provide PAC guarantees for their solutions 3. They are model-specific. In contrast, we propose model-agnostic, sample-efficient methods to solve TkIP/TkIP-O with PAC guarantees.

2.1 SHAP

SHAP (SHapley Additive exPlanations) is based on a game-theoretic concept called Shapley values (Shapley et al. 1953). SHAP applies this concept to explain the predictions of the model by treating individual features as players and the output of the model as the payoff. By measuring the marginal contributions of features across different coalitions, SHAP assigns a score to each feature that reflects its contribution to the final prediction of the model. Given a set of features $D = \{1, 2, .., d\}$, the SHAP value ϕ_i for the i^{th} feature of an input x with a model f is computed by taking the weighted average of the change in predictions of f when feature i is added to a subset of features S as shown in Eqn.1.

$$\phi_i(x, f) = \sum_{S \subseteq D \setminus \{i\}} \frac{|S|!(d - |S| - 1)!}{d!} \left[f(x_{S \cup \{i\}}) - f(x_S) \right] \tag{1}$$

Here x_S is the feature vector restricted to S. To evaluate the model function with missing features in the above expression, we use the interventional SHAP formulation (Chen et al. 2020), where missing feature values are set to a default baseline. Note that computing SHAP values exactly has a computational complexity of $\Theta(2^d)$. To reduce computational costs, several approximation techniques have been proposed. We describe two sampling-based techniques to estimate SHAP values (SamplingSHAP/KernelSHAP) that allow for generating confidence intervals and also provide a brief overview of other efficient techniques that are either model-specific or do not have guarantees on accuracy.

2.2 SamplingSHAP

SamplingSHAP estimates SHAP values by only evaluating a subset of terms in Eqn.1 and then averaging over the resulting marginals. Štrumbelj et al. (Štrumbelj and Kononenko 2014) provide an efficient algorithm to perform Monte Carlo sampling according to the probability distribution induced by the weights in Eqn.1. To quantify the uncertainty in the SHAP estimate based on the number of samples, Merrick et al. (Merrick and Taly 2020) proposed the use of Standard Error of Means (SEM) to derive confidence intervals through the *Central Limit Theorem (CLT)*. Specifically, the Monte Carlo simulation is run T_i times for each feature i, each time yielding an estimate $\hat{\phi}_i^j$, thus giving a set of SHAP estimates $\{\hat{\phi}_i^j\}_{j=1}^{T_i}$. Finally, the SHAP value for i is set to be $\hat{\phi}_i = \sum_{j=1}^{T_i} \hat{\phi}_i^j / T_i$. Eqn.2 shows the 95% CI for the i^{th} feature (there's a 0.95 probability of ϕ_i being in CI_i):

$$CI_i = \left[\hat{\phi}_i \pm 1.96 \frac{\sigma_i}{\sqrt{T_i}}\right]. \tag{2}$$

Here σ_i denotes the standard deviation of the set of SHAP estimates $\{\hat{\phi}_i^j\}_{j=1}^{T_i}$. Note that we can achieve any confidence that we want, by tweaking the parameter 1.96 accordingly.

Additionally, prior works have also tried to reduce the length of the CIs through variance reduction techniques. For instance, Mitchell et al. (Mitchell et al. 2022) propose to evaluate negatively correlated pairs of samples in SamplingSHAP to reduce the variance σ_i of SHAP estimates. Sampling techniques have also been used in the context of Game Theory for computing Shapley values (Maleki et al. 2014; Burgess and Chapman 2021; Castro, Gómez, and Tejada 2009).

2.3 KernelSHAP

KernelSHAP (Lundberg and Lee 2017) is another sampling-based method that views SHAP values as the solution to a weighted regression problem. Specifically, consider a linear model of the form $g(S) = \phi_0 + \sum_{i \in S} \phi_i$, where ϕ_i denote the SHAP values. KernelSHAP proposes to estimate these values by solving the following optimization problem:

$$\{\phi_i\} = \underset{\phi_1,..\phi_d}{\arg\min} \sum_{S \subseteq D} w(S)\Big(f(S) - g(S)\Big)^2. \tag{3}$$

Here, $w(S)$ is a weighting function that is chosen in a way that makes solving Eqn.3 equivalent to finding SHAP values. Note that evaluating Eqn.3 requires evaluating an exponential number of terms in the summation, making the computation of exact SHAP values intractable. Fortunately, an approximation of Eqn.3 that evaluates only a small subset of terms is sufficient in practice to estimate SHAP values. Furthermore, a recent work (Covert and Lee 2020) has shown that the variance of SHAP values, computed by using KernelSHAP, can be used to derive confidence intervals, providing a means of detecting convergence in the SHAP estimates; this leads to CIs identical to those of Eqn.2.

2.4 Other Efficient Estimation Techniques

Several techniques have been developed to more efficiently estimate SHAP values. TreeSHAP was developed to efficiently compute exact SHAP values for tree-based models (Lundberg, Erion, and Lee 2018; Lundberg et al. 2020). A more recent work (Chen et al. 2023) makes modifications to neural networks to efficiently compute SHAP values in a single forward pass using Harsanyi interactions. While these techniques are efficient and accurate, they are model-specific. On the other hand, prior works have proposed model-agnostic methods to efficiently estimate SHAP values. For instance, (Jethani et al. 2021; Covert, Kim, and Lee 2022) directly predict SHAP values using a learned model. However, these methods do not have guarantees on accuracy and may produce incorrect SHAP values, which make them unsuitable for our problem setting.

3 Problem Setting

In this section, we define the Top-k Identification Problem (TkIP) and the Ordered Top-k identification problem (TkIP-O). The goal of TkIP is to identify the features with the highest SHAP values (referred to as the Top-k features). TkIP-O additionally requires the Top-k features to be ordered according to their SHAP value. To apply sampling-based techniques to solve these problems, we define (ϵ, δ)-PAC solutions for both TkIP and TkIP-O. This allows for an ϵ-approximate version of the solution with a low probability of failure (δ). Finally, we describe a naive stopping condition that can be used with Kernel/Sampling-SHAP to derive a (ϵ, δ)-PAC solution for both TkIP and TkIP-O. We demonstrate that this naive solution is sample-inefficient, motivating the need for our proposed solutions that improve sample efficiency.

3.1 Top-k Identification Problem

Consider a model $f : \mathcal{I} \rightarrow \mathbb{R}$, which acts on a d-dimensional input $x \in \mathcal{I}$ to produce a prediction $p = f(x)$. For an input $x \in \mathcal{I}$, let $\{\phi_1, \phi_2, ..., \phi_d\}$ denote the set of SHAP values corresponding to the input features $D = \{1, 2, .., d\}$ respectively. To simplify notation, let us assume that the features are indexed such that:

$$\phi_1 \geq \phi_2 \geq \phi_3 .. \geq \phi_d. \tag{4}$$

The goal of TkIP is to identify the k features: TOPK$= \{1, 2, .., k\}$ corresponding to the k highest SHAP values $\mathcal{S} = \{\phi_1, \phi_2, ..\phi_k\}$. Note that the ordering of features in TOPK does not matter. Solving TkIP exactly requires us to precisely evaluate all the SHAP values, which is computationally intractable. Instead, we define $\epsilon-$*approximate* and *(ϵ, δ)-PAC* solutions for TkIP that are more useful in the context of sampling-based PAC methods.

- **ϵ-approximate solution.** For a given accuracy parameter $\epsilon > 0$, consider a subset of features $D^* \subset D$ such that $|D^*| = k$. D^* is an ϵ-approximate solution to TkIP if it satisfies the following two conditions:

$$Inclusion : \phi_i \geq \phi_k - \epsilon, \forall i \in D^*, \tag{5}$$

$$Exclusion : \phi_j \leq \phi_k + \epsilon, \forall j \in D - D^*. \tag{6}$$

The inclusion condition (Eqn. 5) requires that all the features in the solution (D^*) have a SHAP value of at least $\phi_k - \epsilon$. The exclusion condition (Eqn. 6) ensures that any feature that's not part of the solution does not have a SHAP value that exceeds $\phi_k + \epsilon$.

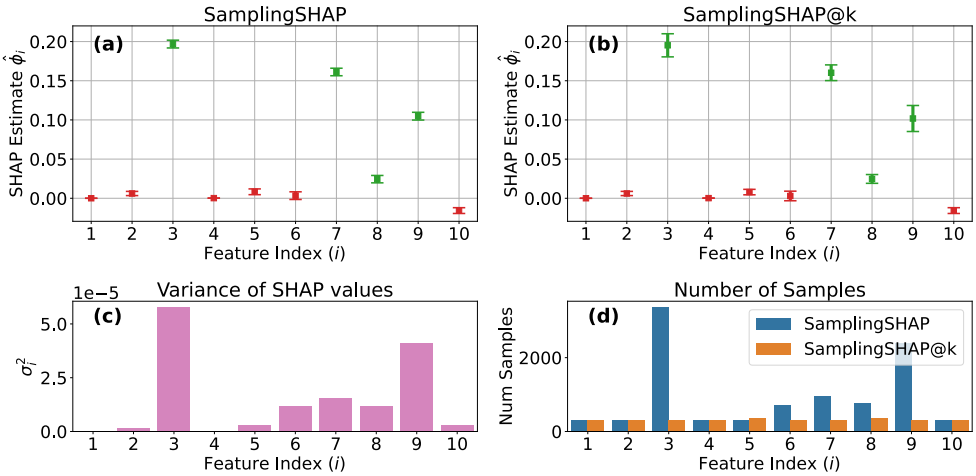

Figure 1: Comparing SamplingSHAP and SamplingSHAP@k for solving TkIP. (a) CIs of SHAP estimates with SamplingSHAP (b) CIs of SHAP estimates with SamplingSHAP@k (c) Variance of SHAP estimates (d) Costs of SamplingSHAP and SamplingSHAP@k. The cost of SamplingSHAP scales with the variance of the SHAP estimates, resulting in high cost. SamplingSHAP improves sample efficiency by using an improved stopping condition and sampling scheme.

- **(ϵ, δ)-PAC solution.** For given accuracy and confidence parameters $\epsilon, \delta \in (0, 1)$, D^* is said to be an (ϵ, δ) solution for TkIP if it is an ϵ-approximate solution with a probability of at least $1 - \delta$:

$$\Pr[Inclusion \wedge Exclusion] \geq 1 - \delta. \tag{7}$$

This relaxed notion of the solution allows for a feature i to be returned as part of the solution even if $i \notin$ TOPK, as long as the corresponding SHAP value ϕ_i is ϵ-close to ϕ_k (i.e. the k^{th} SHAP value).

3.2 Ordered Top-k Identification Problem

The goal of TkIP-O is to find the k highest features, ordered according to their SHAP values. Note that, unlike TkIP, the features in TOPK need to be ordered for TkIP-O. To formalize the notion of approximate solutions for TkIP-O, we start by defining ϵ-tolerant ranked groups of features for D as follows:

$$G_{r,\epsilon} := \{i : |\phi_i - \phi_r| \leq \epsilon\} \tag{8}$$

$G_{r,\epsilon}$ includes all features that are at most ϵ away from the $r - th$ ranked feature. We can now define *$\epsilon-$approximate* and *(ϵ, δ)-PAC* solutions for TkIP-O.

- **ϵ-approximate solution.** For a given accuracy parameter $\epsilon > 0$, consider an ordered subset of features $D^* \subset D$ such that $|D^*| = k$. Let i_r be the r-th feature in the order induced by D^*. D^* is an ϵ-approximate solution to TkIP-O if it satisfies the following:

$$i_r \in G_{r,\epsilon}, \forall i_r \in D^*. \tag{9}$$

- **(ϵ, δ)-PAC solution.** For given accuracy and confidence parameters $\epsilon > 0, \delta \in (0, 1)$, D^* is said to be an (ϵ, δ) solution for TkIP if it is an ϵ-approximate solution with a probability at least $1 - \delta$:

$$\Pr[i_r \in G_{r,\epsilon}, \forall i_r \in D^*] \geq 1 - \delta. \tag{10}$$

Note that the above ϵ approximate solution allows for the order of features to be swapped if their SHAP values are less than a distance ϵ apart. The (ϵ, δ)-PAC solution additionally allows for a small probability of failure (δ).

3.3 PAC Solution for TkIP and TkIP-O With Naive Stopping Condition

In both KernelSHAP and SamplingSHAP, we can use the CLT-based approaches mentioned in Sections 2.2, 2.4 to obtain confidence intervals of the following form. Let ϕ_i be the true SHAP value for feature i, and let $\hat{\phi}_i$ be our approximation for it. Then, if we repeat the corresponding algorithm T_i times, with probability at least $1 - \frac{\delta}{d}$ we have:

$$CI_i = \left[\hat{\phi}_i \pm Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}}\right] \tag{11}$$

In the above, $Z(\delta/d)$ is the critical value from the standard normal distribution for the desired level of confidence; note that this value is a small constant. It is clear from Eqn. 11, that the larger T_i is, the closer our approximation is to the true value. One way to identify the TOPK features is by running the SHAP estimation algorithm (i.e. adding more samples) until the CIs for all the features are small enough to meet the following stopping condition:

$$|CI_i| = 2 \cdot Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}} \leq \epsilon, \forall i \in D. \tag{12}$$

We call this the *naive stopping condition*, and in Theorem 1 we show that it indeed leads to an (ϵ, δ)-PAC solution for TkIP and TkIP-O. Thus, Kernel/Sampling-SHAP can be straightforwardly adapted to solve TkIP by using enough samples to meet this stopping condition. In the following subsection, we will explain why this naive approach is sample-inefficient with the aid of an example, motivating the need for a better stopping condition and sampling technique.

Theorem 1. *Let $\mathcal{S} = \{\hat{\phi}_1, \hat{\phi}_2, .., \hat{\phi}_d\}$ denote the SHAP estimates of input features $D = \{1, 2, .., d\}$, such that the CI_is defined using a confidence of $\frac{\delta}{d}$ satisfy $|CI_i| \leq \epsilon, \forall i \in D$. Then, $D^* = \arg\max_k(\mathcal{S})$ is an (ϵ, δ)-PAC solution for TkIP and TkIP-O; the solution consists of an ordered set of k features with the largest $\hat{\phi}_i$.*

3.4 Inefficiencies of the Naive Stopping Condition

The Naive stopping condition requires the CIs of all the features to be of width at most ϵ. For a feature i, the number of samples N_i necessary to achieve this is proportional to the variance of the feature's SHAP estimate ($N_i \propto \sigma_i^2$), resulting in high-variance features incurring a higher sample-cost. To illustrate, we apply SamplingSHAP to explain the prediction of an MLP model on a single example from the UCI Credit dataset. To identify the Top-k features (with $k = 4$), we obtain CIs by runing SamplingSHAP multiple times for each feature, until the stopping condition in Eqn. 12 is met. We visualize the CIs of the SHAP estimates of the individual features in Fig.1a, where the Top-4 features are marked as green. To understand the cost of this stopping condition, we plot the number of function evaluations consumed by the algorithm in Fig.1d and the variance of the SHAP estimate for each feature in Fig.1c. As expected, we find that the cost is proportional to the variance of the per-feature SHAP estimate, resulting in a high sample-cost for high-variance features.

A key drawback of the naive sampling scheme is that *it requires $|CI_i| \leq \epsilon$ for all features, regardless of the uncertainty that the feature belongs in* TOPK. This results in a lot of wasted samples. For instance, in the example in Fig.1, $\hat{\phi}_3$ (SHAP estimate for $feature$-3) is much higher compared to the other features, allowing us to conclude with high confidence that $3 \in$ TOPK early on in the sampling process and avoid sampling $feature$-3 further. However the naive sampling scheme lacks such adaptivity and forces this high-variance feature to continue sampling until $|CI_3| \leq \epsilon$, thus leading to a lot of wasted samples and contributing significantly to the sample cost of SamplingSHAP. In the next section we develop $SamplingSHAP@k$ and $KernelSHAP@k$ to avoid such wasted samples by using a modified stopping condition and sampling scheme.

4 Our Proposal

We develop efficient methods to solve TkIP and TkIP-O by leveraging ideas from the MAB literature. For solving TkIP, we propose KernelSHAP@k and SamplingSHAP@k by framing TkIP as a variant of the MAB *Explore-m* problem. To solve TkIP-O, our approach involves first identifying the top-k features using TkIP and then solving an ordering problem with the top-k features. We develop KernelSHAP-O and SamplingSHAP-O to efficiently solve the ordering variant. Overall, our methods can be used to efficiently solve TkIP and TkIP-O with PAC guarantees. We describe our proposals in greater detail in the remainder of this section.

4.1 Adapting SamplingSHAP and KernelSHAP to solve TkIP

Explore-m vs. TkIP. TkIP can be viewed as a variant of the *Explore-m* problem in multi-arm bandits, where given d arms with unknown distributions of payoffs, the goal is to select the m-best arms which have the highest expected payoffs. A key difference between *Explore-m* and TkIP is that the ϵ-approximate solution for the *Explore-m* problem typically only requires the *inclusion condition* (Eqn. 5) to be satisfied. In contrast, TkIP additionally requires the satisfaction of the *exclusion condition* (Eqn. 6). Despite this difference in formulation, we show that the LUCB algorithm (Kalyanakrishnan et al. 2012) (which was originally developed to solve the Explore-m problem) can be adapted to solve TkIP. We leverage two key ideas (*M1, M2*) from this algorithm to modify KernelSHAP and SamplingSHAP:

M1. Overlap-based stopping condition. We use the stopping condition described in Theorem 2 that considers the overlap in the CIs for the SHAP estimates. This overlap-based stopping condition avoids the need to reduce all the CI widths to below ϵ as shown in Fig. 1b.

We introduce some notation to formally describe the stopping condition. Let T_i the number of SHAP estimates that we have collected so far for feature i. For the desired confidence δ, we define a δ/d confidence interval $CI_i = [\alpha_i, \beta_i]$ as before, where $\hat{\phi}_i$ the current SHAP estimate, $\alpha_i = \hat{\phi}_i - Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}}$, and $\beta_i = \hat{\phi}_i + Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}}$.

Theorem 2. *Let $High$ denote the set of k features with the highest SHAP estimates $\hat{\phi}_i$ and Low denote the remaining set of $d-k$ features. Let h be the feature in $High$ with the lowest lower confidence bound i.e. $h = \arg\min_{i \in High}\{\alpha_i\}$, and let ℓ be the feature in Low with the highest higher confidence bound i.e. $\ell = \arg\max_{i \in Low}\{\beta_i\}$. Then, $High$ is a (ϵ, δ)-PAC solution for TkIP if the following condition is satisfied: $\beta_\ell - \alpha_h \leq \epsilon$.*

M2. Greedy sampling scheme. The default variance-based sampling scheme used by SamplingSHAP minimizes the CIs for all features. Such sampling schemes are inefficent for the stopping condition in Theorem 2, which only depends on two features (h and ℓ) at any given point in the sampling process. To improve the sample efficiency, we consider a greedy sampling strategy as described in Algorithm 1 (Appendix B). The algorithm identifies h and ℓ as defined in Theorem 2, and evaluates additional SHAP estimates for these two features. These steps are repeated until the stopping condition is met. At this point, $High$ will be an (ϵ, δ)-PAC solution for $TkIP$. This scheme improves sample efficiency by allocating samples only for (h, ℓ), which are exactly the features that affect the stopping condition. To see why this algorithm terminates, notice that in each iteration exactly 2 CIs shrink. Therefore, in the worst case, there will come a point where all CIs will be of length at most ϵ, and thus the stopping condition will trivially be true.

KernelSHAP@k and SamplingSHAP@k. We apply the above changes to existing algorithms to propose KernelSHAP@k (KernelSHAP + M1) and SamplingSHAP@k (SamplingSHAP + M1 + M2). In both cases, we incrementally add SHAP estimates $\hat{\phi}_i^j$ until the stopping condition (M1) is met and the TOPK features are identified. Additionally, for SamplingSHAP@k, we use the more efficient greedy sampling scheme (M2) that allocates samples only to features that influence the stopping condition. Note that the greedy sampling scheme (M2) requires the ability to compute the SHAP values of features individually. Thus, we cannot apply M2 to KernelSHAP as it estimates the SHAP values of all features together.

	Methods	German Credit	Give Me Some Credit	HELOC	UCI Credit
#Samples	KernelSHAP (naive)	1171116	45457	398983	165558
	KernelSHAP@k	96599 ($\downarrow 12.1\times$)	7804 ($\downarrow 5.8\times$)	39610 ($\downarrow 10.0\times$)	21807 ($\downarrow 7.6\times$)
	SamplingSHAP (naive)	2082265	108292	748727	246052
	SamplingSHAP@k	**52047** ($\downarrow 40\times$)	**4328** ($\downarrow 25.0\times$)	**25471** ($\downarrow 29.4\times$)	**16662** ($\downarrow 14.8\times$)
Time (s)	KernelSHAP (naive)	12.61	0.12	1.64	1.15
	KernelSHAP@k	1.032 ($\downarrow 12.21\times$)	0.02 ($\downarrow 6\times$)	0.16 ($\downarrow 10.2\times$)	0.15 ($\downarrow 7.67\times$)
	SamplingSHAP (naive)	3.49	0.16	1.19	0.42
	SamplingSHAP@k	**0.09** ($\downarrow 38.8\times$)	**0.006** ($\downarrow 26.7\times$)	**0.041** ($\downarrow 29.02\times$)	**0.03** ($\downarrow 14\times$)

Table 1: Sample cost and runtime required to find an $(\epsilon = 0.005, \delta = 10^{-6})$-PAC solution for TkIP ($k = 4$) using different methods. Our methods are significantly more efficient compared to the baselines with the naive stopping condition.

4.2 Approach for Solving TkIP-O

To solve TkIP-O, we start by noting that it is possible to reduce it to the *ordering problem*. In MAB the ordering problem involves ordering d arms according to their expected payoffs. One way to apply this to TkIP-O is to solve the ordering problem for all d features and return the *top-k* features. This direct application of the MAB ordering problem is sample-inefficient as it unnecessarily orders all features. Instead, we take a two-step approach, where we first isolate the top-k features by solving TkIP (P1) using Kernel/SamplingSHAP@k and then solve the ordering problem, just on the top-k features (P2). We show in Theorem 3 that this two-step approach indeed provides a PAC solution for TkIP-O.

Theorem 3. *Let D^* be an (ϵ, δ)-PAC solution for TkIP. Let $D^\dagger$ be an (ϵ, δ)-PAC solution to the ordering problem applied on D^*. Then $D^\dagger$ is a $(\epsilon, 2\delta)$-PAC solution for TkIP-O.*

4.3 Adapting SamplingSHAP and KernelSHAP To Solve the Ordering Problem

To solve the ordering problem in the context of SHAP values, we leverage two key ideas based on the LUCBRank (Kataria et al. 2018) algorithm to adapt Kernel/SamplingSHAP:

M3. Pairwise overlap-based stopping condition. We use a stopping condition that checks the pairwise overlap in CIs for every pair of successive ordered features to decide when the ordering problem has been solved. We introduce some notation to formally state the stopping condition. Let $D = \{1, 2, .., d\}$ denote the set of features that we want to order. Let $D^\dagger$ represent the current ordering of these features, with $D^\dagger_r$ denoting the feature in the r^{th} position. For the desired confidence δ, we define a δ/d confidence interval $CI_i = [\alpha_i, \beta_i]$ as before, where $\hat{\phi}_i$ the current SHAP estimate, $\alpha_i = \hat{\phi}_i - Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}}$ and $\beta_i = \hat{\phi}_i + Z(\delta/d)\frac{\sigma_i}{\sqrt{T_i}}$. (Katariya et al. 2018) show that $D^\dagger$ is an (ϵ, δ)-PAC solution if the following stopping condition is met:

$$\beta_{D^\dagger_r} - \alpha_{D^\dagger_{r+1}} \leq \epsilon, \forall r \in [1, d-1]. \tag{13}$$

M4. Greedy sampling scheme for ordering. For SamplingSHAP, we use a greedy sampling scheme as described in Algorithm 2 (in Appendix B). This scheme improves sample efficiency by only sampling features for which the stopping condition hasn't been met in each round. Note that this sampling scheme is incompatible with KernelSHAP as it requires the ability to sample SHAP values at a feature-level.

KernelSHAP-O and SamplingSHAP-O. By incorporating the above modifications, we develop KernelSHAP-O (KernelSHAP+M3) and SamplingSHAP-O (SamplingSHAP+M3+M4). These methods provide a sample-efficient way to solve the ordering problem. By using these methods in composition with Kernel/SamplingSHAP@k, we can obtain PAC solutions for TkIP-O.

Datasets	# Feat	# Train	#Test
German Credit (Hofmann 2013)	61	800	56
GMSC (Credit Fusion 2011)	11	96215	100
HELOC (FICO 2018)	23	8367	100
UCI Credit (Quinlan 1987)	15	522	66

Table 2: Datasets used in our experiments.

5 Experiments

In this section, we perform evaluations to quantify the sample-cost and runtime improvements of our proposed methods for solving TkIP and TkIP-O.

5.1 Experimental Setup

Since there are no prior works that provide PAC solutions for TkIP/TkIP-O, we use Kernel/SamplingSHAP with the naive stopping condition as our baseline. Note that this baseline returns an ordered list of top-k features and thus can be treated as an (ϵ, δ)-PAC solution for TkIP and TkIP-O. For TkIP, we compare the performance of Kernel/SamplingSHAP@k with that of their naive baseline counterparts. To solve TkIP-O, we first solve TkIP and then solve the ordering problem with the top-k features. We evaluate two different combinations of our proposed methods: KernelSHAP@k + KernelSHAP-O and SamplingSHAP@k + SamplingSHAP-O. Note that for TkIP-O, the failure probability (δ) is split evenly between TkIP and the ordering problem. Additionally, for TkIP-O, we compare our proposed two-step approach against a direct application of the ordering problem by ordering all d features and returning the *top-k* features. We compare the performance of various methods in terms of sample-cost (i.e. number of function f evaluations) and runtime.

Table 2 lists the datasets used in our experiments, along with a brief description of the prediction task, number of

	Methods	German Credit	Give Me Some Credit	HELOC	UCI Credit
#Samples	KernelSHAP (naive)	1171116	45457	398983	165558
	KernelSHAP-O (direct)	670254 ($\downarrow 1.74\times$)	20062 ($\downarrow 2.26\times$)	214591 ($\downarrow 1.85\times$)	89391 ($\downarrow 1.85\times$)
	KernelSHAP@k+KernelSHAP-O	195334 ($\downarrow 6\times$)	9074 ($\downarrow 5\times$)	72972 ($\downarrow 5.5\times$)	25466 ($\downarrow 6.5\times$)
	SamplingSHAP (naive)	2082265	108292	748727	246052
	SamplingSHAP-O (direct)	297626 ($\downarrow 7\times$)	10838 ($\downarrow 10\times$)	132560 ($\downarrow 5.6\times$)	26758 ($\downarrow 9.2\times$)
	SamplingSHAP@k+SamplingSHAP-O	**101862** ($\downarrow 20.4\times$)	**8173** ($\downarrow 13.2\times$)	**55933** ($\downarrow 13.4\times$)	**19178** ($\downarrow 12.8\times$)
Time (s)	KernelSHAP (naive)	12.61	0.12	1.64	1.15
	KernelSHAP-O (direct)	7.07 ($\downarrow 1.8\times$)	0.05 ($\downarrow 2.4\times$)	0.8 ($\downarrow 2.05\times$)	0.59 ($\downarrow 1.95\times$)
	KernelSHAP@k+KernelSHAP-O	2.04 ($\downarrow 6.2\times$)	0.024 ($\downarrow 5\times$)	0.28 ($\downarrow 5.8\times$)	0.17 ($\downarrow 6.7\times$)
	SamplingSHAP (naive)	3.49	0.16	1.19	0.42
	SamplingSHAP-O (direct)	0.53 ($\downarrow 6.6\times$)	0.016 ($\downarrow 10\times$)	0.2 ($\downarrow 5.95\times$)	0.04 ($\downarrow 10.5\times$)
	SamplingSHAP@k+SamplingSHAP-O	**0.18** ($\downarrow 19.4\times$)	**0.012** ($\downarrow 13.3\times$)	**0.087** ($\downarrow 13.7\times$)	**0.032** ($\downarrow 13.1\times$)

Table 3: Sample cost and runtimes required for finding $(\epsilon = 0.005, \delta = 10^{-6})$-PAC solution for TkIP-O ($k = 4$) using different methods. Our proposed two-step methods (Kernel/SamplingSHAP-O@k + Kernel/SamplingSHAP-O) have significantly better runtime and sample cost compared to the naive baseline and direct application of the ordering algorithms.

features, and train/test split. In each case, we train a 5-layer MLP model on the binary classification task using the training set for 100 epochs, and use this model to make predictions on the test set. For the negatively classified examples in the test set (indicating a high likelihood of the credit application being rejected), we use different methods to identify the Top-4 features (i.e. $k = 4$) that contributed the most to the negative prediction in terms of their SHAP values. We use interventional SHAP for our experiments and use a positively classified example from the training set as our baseline.

5.2 Results

TkIP. Table 1 compares the average sample cost and runtime required by different methods to solve TkIP with a $(\epsilon = 0.005, \delta = 10^{-6})$-PAC guarantee across different datasets. Our evaluations show that Kernel/SamplingSHAP@k significantly outperform their baseline counterparts Kernel/SamplingSHAP (naive), offering improvements of $5.8\times - 40\times$ in sample efficiency and $6\times - 38.8\times$ in runtime. Furthermore, we find that SamplingSHAP@k has a consistently lower sample-cost and runtime compared to kernelSHAP@k. Interestingly, even in cases where the two methods have comparable sample-costs, SamplingSHAP@k has a significantly lower runtime. E.g. for the UCI credit dataset, the sample cost of SamplingSHAP@k compared to KernelSHAP@k improves by $1.3\times$, whereas runtime improves by $5\times$. The reason is that each KernelSHAP estimate is expensive to compute as it requires solving a weighted regression problem. In contrast, SamplingSHAP works by just computing a simple averages, which requires much less compute, resulting in a faster runtime.

TkIP-O. Table 3 compares different methods for solving TkIP-O. Our proposed methods offer significant improvements of $5\times - 20.4\times$ in sample cost and $5\times - 19.4\times$ in runtime compared to the naive baseline. Our methods also outperform the direct application of the ordering problem to solve TkIP, highlighting the benefit of our proposed two-step approach. As before, SamplingSHAP@k+SamplingSHAP-O outperforms KernelSHAP@k+KernelSHAP-O in terms of both sample cost and runtime.

5.3 Sensitivity Studies

To understand how the accuracy parameter ϵ influences the sample-efficiency of various methods, we perform sensitivity studies by varying ϵ between $[0.005, 0.01]$. For different values of ϵ, we plot the sample cost and runtime of different methods for solving TkIP across the four datasets considered in our experiments. Due to space constraints, the results for this study are presented in Appendix C. In general, we find that the trends in Table 1 continue to hold across different values of ϵ. We refer the reader to Appendix C for a more detailed discussion of these results.

6 Limitations

The methods we propose inherently make the assumption that the sampling-based technique used to estimate SHAP values are unbiased. While this is true for SamplingSHAP, KernelSHAP does not satisfy this property as it can be biased. However, prior work has shown this bias to be negligibly small (Covert and Lee 2020) and also developed unbiased versions of KernelSHAP. It is important for any practical applications of our methods to be aware of such sources of bias that can interfere with PAC guarantees.

7 Conclusion

Motivated by applications in finance, we introduce and study the *Top-k Identification problem (TkIP)* and *Ordered Top-k Identificaiton problem (TkIP-O)*, where the goal is to identify an unordered/ordered set of k features that have the highest SHAP values. We leverage connections with multi-arm bandits to solve these problems efficiently. We show that TkIP can be reduced to a stricter variant of the *Explore-m problem* and TkIP-O can be viewed as a composition of TkIP and an ordering problem. This framing allows us to leverage algorithms from MAB literature to develop sample-efficient solutions. We develop Kernel/SamplingSHAP@k to solve TkIP and Kernel/SamplingSHAP-O to solve the ordering problem in TkIP-O. Experiments on credit-related datasets show that our proposed methods offer significant improvements in sample cost (up to $40\times$) and runtime (up to $39\times$).

Acknowledgements

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("J.P. Morgan") and is not a product of the Research Department of J.P. Morgan. J.P. Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Bhatt, U.; Xiang, A.; Sharma, S.; Weller, A.; Taly, A.; Jia, Y.; Ghosh, J.; Puri, R.; Moura, J. M. F.; and Eckersley, P. 2020. Explainable machine learning in deployment. In Hildebrandt, M.; Castillo, C.; Celis, L. E.; Ruggieri, S.; Taylor, L.; and Zanfir-Fortuna, G., eds., *FAT* '20: Conference on Fairness, Accountability, and Transparency, Barcelona, Spain, January 27-30, 2020*, 648–657. ACM.

Burgess, M. A.; and Chapman, A. C. 2021. Approximating the Shapley Value Using Stratified Empirical Bernstein Sampling. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 73–81. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Castro, J.; Gómez, D.; and Tejada, J. 2009. Polynomial Calculation of the Shapley Value Based on Sampling. *Comput. Oper. Res.*, 36(5): 1726–1730.

Chen, H.; Janizek, J. D.; Lundberg, S.; and Lee, S.-I. 2020. True to the Model or True to the Data?

Chen, L.; Lou, S.; Zhang, K.; Huang, J.; and Zhang, Q. 2023. HarsanyiNet: Computing Accurate Shapley Values in a Single Forward Propagation. *arXiv preprint arXiv:2304.01811*.

Congress, U. 1970. Fair Credit Reporting Act. Public Law 91-508. 15 U.S.C. §1681 et seq.

Congress, U. 1974. Equal Credit Opportunity Act. Public Law 93-495. 15 U.S.C. §1691 et seq.

Covert, I.; Kim, C.; and Lee, S.-I. 2022. Learning to estimate shapley values with vision transformers. *arXiv preprint arXiv:2206.05282*.

Covert, I.; and Lee, S.-I. 2020. Improving kernelshap: Practical shapley value estimation via linear regression. *arXiv preprint arXiv:2012.01536*.

Credit Fusion, W. C. 2011. Give Me Some Credit.

FICO. 2018. Explainable machine learning challenge, 2018. https://community.fico.com/s/explainable-machine-learning-challenge.

Hofmann, H. 2013. UCI machine learning repository: Statlog (german credit data) data set.

Jethani, N.; Sudarshan, M.; Covert, I. C.; Lee, S.-I.; and Ranganath, R. 2021. Fastshap: Real-time shapley value estimation. In *International Conference on Learning Representations*.

Kalyanakrishnan, S.; Tewari, A.; Auer, P.; and Stone, P. 2012. PAC subset selection in stochastic multi-armed bandits. In *ICML*, volume 12, 655–662.

Katariya, S.; Jain, L.; Sengupta, N.; Evans, J.; and Nowak, R. 2018. Adaptive sampling for coarse ranking. In *International Conference on Artificial Intelligence and Statistics*, 1839–1848. PMLR.

Kolpaczki, P.; Bengs, V.; and Hüllermeier, E. 2021. Identifying Top-k Players in Cooperative Games via Shapley Bandits. In *LWDA*, 133–144.

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

Lundberg, S. M.; Erion, G. G.; and Lee, S.-I. 2018. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*.

Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

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

Merrick, L.; and Taly, A. 2020. The explanation game: Explaining machine learning models using shapley values. In *Machine Learning and Knowledge Extraction: 4th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2020, Dublin, Ireland, August 25–28, 2020, Proceedings 4*, 17–38. Springer.

Mitchell, R.; Cooper, J.; Frank, E.; and Holmes, G. 2022. Sampling permutations for shapley value estimation.

Quinlan, J. 1987. Credit Approval. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5FS30.

Shapley, L. S.; et al. 1953. A value for n-person games.

Štrumbelj, E.; and Kononenko, I. 2014. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41: 647–665.