Efficient Task-Specific Data Valuation for Nearest Neighbor Algorithms

Ruoxi Jia[1] David Dao[2] Boxin Wang[3] Frances Ann Hubis[2] Nezihe Merve Gurel[2]
Bo Li[4] Ce Zhang[2] Costas Spanos[1] Dawn Song[1]

[1]UC Berkeley [2]ETH Zurich [3]Zhejiang University [4]UIUC

ruoxijia@berkeley.edu, daviddao@inf.ethz.ch, boxin.wang@outlook.com, hubisf@inf.ethz.ch, nezihe.guerel@inf.ethz.ch, lxbosky@gmail.com, ce.zhang@inf.ethz.ch, spanos@berkeley.edu, dawnsong@gmail.com

ABSTRACT

Given a data set $\mathcal{D}$ containing millions of data points and a data consumer who is willing to pay for $\$X$ to train a machine learning (ML) model over $\mathcal{D}$, *how should we distribute this $\$X$ to each data point to reflect its "value"?* In this paper, we define the "relative value of data" via the Shapley value, as it uniquely possesses properties with appealing real-world interpretations, such as fairness, rationality and decentralizability. For general, bounded utility functions, the Shapley value is known to be challenging to compute: to get Shapley values for all N data points, it requires $O(2^N)$ model evaluations for exact computation and $O(N \log N)$ for (ϵ, δ)-approximation.

In this paper, we focus on one popular family of ML models relying on K-nearest neighbors (KNN). The most surprising result is that for unweighted KNN classifiers and regressors, the Shapley value of all N data points can be computed, *exactly*, in $O(N \log N)$ time – an exponential improvement on computational complexity! Moreover, for (ϵ, δ)-approximation, we are able to develop an algorithm based on Locality Sensitive Hashing (LSH) with only *sublinear* complexity $O(N^{h(\epsilon,K)} \log N)$ when ϵ is not too small and K is not too large. We empirically evaluate our algorithms on up to 10 million data points and even our *exact* algorithm is up to three orders of magnitude faster than the baseline approximation algorithm. The LSH-based approximation algorithm can accelerate the value calculation process even further.

We then extend our algorithm to other scenarios such as (1) weighed KNN classifiers, (2) different data points are clustered by different *data curators*, and (3) there are *data analysts* providing computation who also requires proper valuation. *Some* of these extensions, although also being improved exponentially, are less practical for exact computation (e.g., $O(N^K)$ complexity for weigthed KNN). We thus propose an Monte Carlo approximation algorithm, which is $O(N(\log N)^2/(\log K)^2)$ times more efficient than the baseline approximation algorithm.

PVLDB Reference Format:
R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gurel, B. Li, C. Zhang, C. Spanos, D. Song. Efficient Task-Specific Data Valuation for Nearest Neighbor Algorithms. *PVLDB*, 12(11): 1610-1623, 2019.
DOI: https://doi.org/10.14778/3342263.3342637

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342637

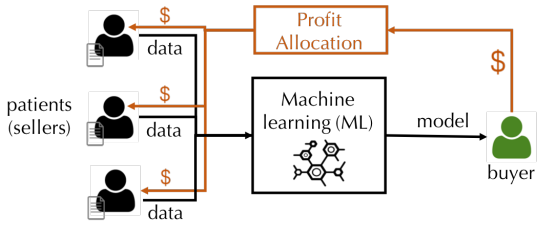

Figure 1: Motivating Example of Data Valuation.

1. INTRODUCTION

"*Data is the new oil*" — large-scale, high-quality datasets are an enabler for business and scientific discovery and recent years have witnessed the commoditization of data. In fact, there are not only marketplaces providing access to data, e.g., IOTA [4], DAWEX [2], Xignite [5], but also marketplaces charging for running (relational) queries over the data, e.g., Google BigQuery [3]. Many researchers start to envision marketplaces for ML models [12].

Data commoditization is highly likely to continue and not surprisingly, it starts to attract interests from the database community. One series of seminal work is conducted by Koutris et al. [33, 34] who systematically studied the theory and practice of "query pricing," the problem of attaching value to running relational queries over data. Recently, Chen et al. [11, 12] discussed "model pricing", the problem of valuing ML models. This paper is inspired by the prior work on query and model pricing, but focuses on a different scenario. In many real-world applications, the datasets that support queries and ML are often contributed by *multiple individuals*. One example is that complex ML tasks such as chatbot training often relies on massive crowdsourcing efforts. A critical challenge for building a data marketplace is thus to allocate the revenue generated from queries and ML models fairly between different data contributors. In this paper, we ask: *How can we attach value to every single data point in relative terms, with respect to a specific ML model trained over the whole dataset?*

Apart from being inspired by recent research, this paper is also motivated by our current effort in building a data market based on privacy-preserving machine learning [14, 28] and an ongoing clinical trial at the Stanford Hospital, as illustrated in Figure 1. In this clinical trial, each patient uploads their encrypted medical record (one "data point") onto a blockchain-backed data store. A "data consumer", or "buyer", chooses a subset of patients (selected according to some non-sensitive information that is not encrypted) and trains a ML model. The buyer pays a certain amount of money that will be distributed back to each patient. In this paper, we focus on the data valuation problem that is abstracted from this real use case and propose novel, practical algorithms for this problem.

Figure 2: Time complexity for computing the SV for KNN models. N is the total number of training data points. M is the number of data contributors. $h(\epsilon, K) < 1$ if $K^* = \max\{1/\epsilon, K\} < C$ for some dataset-dependent constant C.

	Exact	Approximate
Baseline	$2^N N \log N$	$\frac{N^2}{\epsilon^2} \log N \log \frac{N}{\delta}$
Unweighted KNN classifier	$N \log N$	$N^{h(\epsilon,K)} \log N \log \frac{K^*}{\delta}$
Unweighted KNN regression	$N \log N$	—
Weighted KNN	N^K	$\frac{N}{\epsilon^2} \log K \log \frac{K}{\delta}$
Multiple-data-per-curator KNN	M^K	$\frac{N}{\epsilon^2} \log K \log \frac{K}{\delta}$

Specifically, we focus on the *Shapley value* (SV), arguably one of the most popular way of revenue sharing. It has been applied to various applications, such as power grids [9], supply chains [8], cloud computing [49], among others. The reason for its wide adoption is that the SV defines a *unique* profit allocation scheme that satisfies a set of appealing properties, such as fairness, rationality, and decentralizability. Specifically, let $\mathcal{D} = \{z_1, ..., z_N\}$ be N data points and $\nu(\cdot)$ be the "utility" of the ML model trained over a subset of the data points; the SV of a given data point z_i is

$$s_i = \frac{1}{N} \sum_{S \subseteq \mathcal{D} \setminus z_i} \frac{1}{\binom{N-1}{|S|}} \big[\nu(S \cup \{z_i\}) - \nu(S)\big] \tag{1}$$

Intuitively, the SV measures the marginal improvement of utility attributed to the data point z_i, averaged over all possible subsets of data points. Calculating exact SVs requires exponentially many utility evaluations. This poses a radical challenge to using the SV for data valuation–*how can we compute the SV efficiently and scale to millions or even billions of data points?* This scale is rare to the previous applications of the SV but is not uncommon for real-world data valuation tasks.

To tackle this challenge, we focus on a specific family of ML models which restrict the class of utility functions $\nu(\cdot)$ that we consider. Specifically, we study K-nearest neighbors (KNN) classifiers [19], a simple yet popular supervised learning method used in image recognition [25], recommendation systems [6], healthcare [38], etc. Given a test set, we focus on a natural utility function, called the *KNN utility*, which, intuitively, measures the boost of the likelihood that KNN assigns the correct label to each test data point. When $K = 1$, this utility is the same as the test accuracy. Although some of our techniques also apply to a broader class of utility functions (See Section 4), the KNN utility is our main focus.

The contribution of this work is a collection of novel algorithms for efficient data valuation within the above scope. Figure 2 summarizes our technical results. Specifically, we made four technical contributions:

Contribution 1: Data Valuation for KNN Classifiers. The main challenge of adopting the SV for data valuation is its computational complexity — for general, bounded utility functions, calculating the SV requires $O(2^N)$ utility evaluations for N data points. Even getting an (ϵ, δ)-approximation (error bounded by ϵ with probability at least $1 - \delta$) for all data points requires $O(N \log N)$ utility evaluations using state-of-the-art methods (See Section 2.2). For the KNN utility, each utility evaluation requires to sort the training data, which has asymptotic complexity $O(N \log N)$.

C1.1 Exact Computation We first propose a novel algorithm specifically designed for KNN classifiers. We observe that the KNN utility satisfies what we call the *piecewise utility difference* property: the difference in the marginal contribution of two data points z_i and z_j over has a "piecewise form" (See Section 3.1):

$$U(S \cup \{z_i\}) - U(S \cup \{z_j\}) = \sum_{t=1}^{T} C_{i,j}^{(t)} \mathbb{1}[S \in \mathcal{S}_t], \forall S \in \mathcal{D} \setminus \{z_i, z_j\}$$

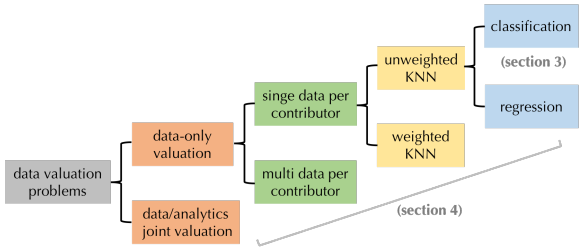

Figure 3: Classification of data valuation problems.

where $\mathcal{S}_t \subseteq 2^{\mathcal{D} \setminus \{z_i, z_j\}}$ and $C_{i,j}^{(t)} \in \mathbb{R}$. This combinatorial structure allows us to design a very efficient algorithm that only has $O(N \log N)$ complexity for *exact* computation of SVs on all N data points. This is an exponential improvement over the $O(2^N N \log N)$ baseline!

C1.2 Sublinear Approximation The exact computation requires to sort the entire training set for each test point, thus becoming time-consuming for large and high-dimensional datasets. Moreover, in some applications such as document retrieval, test points could arrive sequentially and the values of each training point needs to get updated and accumulated on the fly, which makes it impossible to complete sorting offline. Thus, we investigate whether higher efficiency can be achieved by finding approximate SVs instead. We study the problem of getting (ϵ, δ)-approximation of the SVs for the KNN utility. This happens to be reducible to the problem of answering approximate $\max\{K, 1/\epsilon\}$-nearest neighbor queries with probability $1 - \delta$. We designed a novel algorithm by taking advantage of LSH, which only requires $O(N^{h(\epsilon,K)} \log N)$ computation where $h(\epsilon, K)$ is dataset-dependent and typically less than 1 when ϵ is not too small and K is not too large.

Limitation of LSH The $h(\epsilon, K)$ term monotonically increases with $\max\{\frac{1}{\epsilon}, K\}$. In experiments, we found that the LSH can handle mild error requirements (e.g., $\epsilon = 0.1$) but appears to be less efficient than the exact calculation algorithm for stringent error requirements. Moreover, we can extend the exact algorithm to cope with KNN regressors and other scenarios detailed in Contribution 2; however, the application of the LSH-based approximation is still confined to the classification case.

To our best knowledge, the above results are one of the very first studies of efficient SV evaluation designed specifically for utilities arising from ML applications.

Contribution 2: Extensions. Our second contribution is to extend our results to different settings beyond a standard KNN classifier and the KNN utility (Section 4). Specifically, we studied:

C2.1 Unweighted KNN regressors.

C2.2 Weighted KNN classifiers and regressors.

C2.3 One "data curator" contributes multiple data points *and* has the freedom to delete all data points at the same time.

C2.4 One "data analyst" provides ML analytics and the system attaches value to both the analyst and data curators.

The connection between different settings are illustrated in Figure 3, where each vertical layer represents a different slicing to the data valuation problem. In some of these scenarios, we successfully designed algorithms that are as efficient as the one for KNN classifiers. In some other cases, including weigthed KNN and the multiple-data-per-curator setup, the exact computation algorithm is less practical although being improved exponentially.

Contribution 3: Improved Monte Carlo Approximation for KNN. To further improve the efficiency in the less efficient cases, we strengthen the sample complexity bound of the state-of-the-art approximation algorithm, achieving an $\mathcal{O}(N \log^2 N / \log^2 K)$

complexity improvement over the state-of-the-art. Our algorithm requires in total $\mathcal{O}(N/\epsilon^2 \log^2 K)$ computation and is often practical for reasonable ϵ.

Contribution 4: Implementation and Evaluation. We implement our algorithms and evaluate them on datasets up to ten million data points. We observe that our exact SV calculation algorithm can provide up to three orders of magnitude speed-up over the state-of-the-art Monte Carlo approximation approach. With the LSH-based approximation method, we can accelerate the SV calculation even further by allowing approximation errors. The actual performance improvement of the LSH-based method over the exact algorithm depends the dataset as well as the error requirements. For instance, on a *10M subset* of the Yahoo Flickr Creative Commons 100M dataset, we observe that the LSH-based method can bring another $4.6\times$ speed-up.

Moreover, to our best knowledge, this work is also one of the first papers to evaluate data valuation at scale. We make our datasets publicly available and document our evaluation methodology in details, with the hope to facilitate future research on data valuation.

Relationship with Our Previous Work. Unlike this work which focuses on KNN, our previous work [29] considered some generic properties of ML models, such as boundedness of the utility functions, stability of the learning algorithms, etc, and studied their implications for computing the SV. Also, the algorithms presented in our previous work only produce approximation to the SV. When the desired approximation error is small, these algorithms may still incur considerable computational costs, thus not able to scale up to large datasets. In contrast, this paper presents a scalable algorithm that can calculate the exact SV for KNN.

The rest of this paper is organized as follows. We provide background information in Section 2, and present our efficient algorithms for KNN classifiers in Section 3. We discuss the extensions in Section 4 and propose a Monte Carlo approximation algorithm in Section 5, which significantly boosts the efficiency for the extensions that have less practical exact algorithms. We evaluate our approach in Section 6. We discuss the integration with real-world applications in Section 7 and present a survey of related work in Section 8. Due to the space limit, we leave the appendix to the arXiv version of the paper.

2. PRELIMINARIES

We present the setup of the data marketplace and introduce the framework for data valuation based on the SV. We then discuss a baseline algorithm to compute the SV.

2.1 Data Valuation based on the SV

We consider two types of agents that interact in a data marketplace: the sellers (or data curators) and the buyer. Sellers provide training data instances, each of which is a pair of a feature vector and the corresponding label. The buyer is interested in analyzing the training dataset aggregated from various sellers and producing an ML model, which can predict the labels for unseen features. The buyer pays a certain amount of money which depends on the utility of the ML model. Our goal is to distribute the payment fairly between the sellers. A natural way to tackle the question of revenue allocation is to view ML as a cooperative game and model each seller as a player. This game-theoretic viewpoint allows us to formally characterize the "power" of each seller and in turn determine their deserved share of the revenue. For ease of exposition, we assume that each seller contributes one data instance in the training set; later in Section 4, we will discuss the extension to the case where a seller contributes multiple data instances.

Cooperative game theory studies the behaviors of coalitions formed by game players. Formally, a cooperative game is defined by a pair (I, ν), where $I = \{1, \dots, N\}$ denotes the set of all players and $\nu : 2^N \to \mathbb{R}$ is the utility function, which maps each possible coalition to a real number that describes the utility of a coalition, i.e., how much collective payoff a set of players can gain by forming the coalition. One of the fundamental questions in cooperative game theory is to characterize how important each player is to the overall cooperation. The SV [46] is a classic method to distribute the total gains generated by the coalition of all players. The SV of player i with respect to the utility function ν is defined as the average marginal contribution of i to coalition S over all $S \subseteq I \setminus \{i\}$:

$$s(\nu, i) = \frac{1}{N} \sum_{S \subseteq I \setminus \{i\}} \frac{1}{\binom{N-1}{|S|}} \big[\nu(S \cup \{i\}) - \nu(S)\big] \tag{2}$$

We suppress the dependency on ν when the utility is self-evident and use s_i to represent the value allocated to player i.

The formula in (2) can also be stated in the equivalent form:

$$s_i = \frac{1}{N!} \sum_{\pi \in \Pi(I)} \big[\nu(P_i^\pi \cup \{i\}) - \nu(P_i^\pi)\big] \tag{3}$$

where $\pi \in \Pi(I)$ is a permutation of players and P_i^π is the set of players which precede player i in π. Intuitively, imagine all players join a coalition in a random order, and that every player i who has joined receives the marginal contribution that his participation would bring to those already in the coalition. To calculate s_i, we average these contributions over all the possible orders.

Transforming these game theory concepts to data valuation, one can think of the players as training data instances and the utility function $\nu(S)$ as a performance measure of the model trained on the set of training data S. The SV of each training point thus measures its importance to learning a performant ML model. The following desirable properties that the SV *uniquely* possesses motivate us to adopt it for data valuation.

1. **Group Rationality**: The value of the entire training dataset is completely distributed among all sellers, i.e., $\nu(I) = \sum_{i \in I} s_i$.
2. **Fairness**: (1) Two sellers who are identical with respect to what they contribute to a dataset's utility should have the same value. That is, if seller i and j are equivalent in the sense that $\nu(S \cup \{i\}) = \nu(S \cup \{j\}), \forall S \subseteq I \setminus \{i, j\}$, then $s_i = s_j$. (2) Sellers with zero marginal contributions to all subsets of the dataset receive zero payoff, i.e., $s_i = 0$ if $\nu(S \cup \{i\}) = 0$ for all $S \subseteq I \setminus \{i\}$.
3. **Additivity**: The values under multiple utilities sum up to the value under a utility that is the sum of all these utilities: $s(\nu_1, i) + s(\nu_2, i) = s(\nu_1 + \nu_2, i)$ for $i \in I$.

The *group rationality* property states that any rational group of sellers would expect to distribute the full yield of their coalition. The *fairness* property requires that the names of the sellers play no role in determining the value, which should be sensitive only to how the utility function responds to the presence of a seller. The *additivity* property facilitates efficient value calculation when the ML model is used for multiple applications, each of which is associated with a specific utility function. With additivity, one can decompose a given utility function into an arbitrary sum of utility functions and compute value shares separately, resulting in transparency and decentralizability. The fact that the SV is the only value division scheme that meets these desirable criteria, combined with its flexibility to support different utility functions, leads us to employ the SV to attribute the total gains generated from a dataset to each seller.

In addition to its theoretical soundness, our previous work [29] empirically demonstrated that the SV also coincides with people's intuition of data value. For instance, noisy images tend to have lower SVs than the high-fidelity ones; the training data whose distribution is closer to the test data distribution tends to have higher SVs. These empirical results further back up the use of the SV for data valuation. For more details, we refer the readers to [29].

2.2 A Baseline Algorithm

One challenge of applying SV is its computational complexity. Evaluating the exact SV using Eq. (2) involves computing the marginal utility of every user to every coalition, which is $\mathcal{O}(2^N)$. Such exponential computation is clearly impractical for valuating a large number of training points. Even worse, in many ML tasks, evaluating the utility function *per se* (e.g., testing accuracy) is computationally expensive as it requires training a ML model. For large datasets, the only feasible approach currently in the literature is Monte Carlo (MC) sampling [40]. In this paper, we will use it as a baseline for evaluation.

The central idea behind the baseline algorithm is to regard the SV definition in (3) as the expectation of a training instance's marginal contribution over a random permutation and then use the sample mean to approximate it. More specifically, let π be a random permutation of I and each permutation has a probability of $1/N!$. Consider the random variable $\phi_i = \nu(P_i^\pi \cup \{i\}) - \nu(P_i^\pi)$. By (3), the SV s_i is equal to $\mathbb{E}[\phi_i]$. Thus, $\hat{s}_i = \frac{1}{T}\sum_{t=1}^{T} \nu(P_i^{\pi_t} \cup \{i\}) - \nu(P_i^{\pi_t})$ is a consistent estimator of s_i, where π_t be tth sample permutation uniformly drawn from all possible permutations $\Pi(I)$.

We say that $\hat{s} \in \mathbb{R}^N$ is an (ϵ, δ)-approximation to the true SV $s = [s_1, \cdots, s_N]^T \in \mathbb{R}^N$ if $P[\max_i |\hat{s}_i - s_i| \leq \epsilon] \geq 1 - \delta$. Let r be the range of utility differences ϕ_i. By applying the Hoeffding's inequality, [41] shows that for general, bounded utility functions, the number of permutations T needed to achieve an (ϵ, δ)-approximation is $\frac{r^2}{2\epsilon^2} \log \frac{2N}{\delta}$. For each permutation, the baseline algorithm evaluates the utility function for N times in order to compute the SV for N training instances; therefore, the total utility evaluations involved in the baseline approach is $\mathcal{O}(N \log N)$. In general, evaluating $\nu(S)$ in the ML context requires to re-train the model on the subset S of the training data. Therefore, despite its improvements over the exact SV calculation, the baseline algorithm is not efficient for large datasets.

Take the KNN classifier as an example and assume that $\nu(\cdot)$ represents the testing accuracy of the classifier. Then, evaluating $\nu(S)$ needs to sort the training data in S according to their distances to the test point, which has $\mathcal{O}(|S| \log |S|)$ complexity. Since on average $|S| = N/2$, the asymptotic complexity of calculating the SV for a KNN classifier via the baseline algorithm is $\mathcal{O}(N^2 \log^2 N)$, which is prohibitive for large-scale datasets. In the sequel, we will show that it is indeed possible to develop much more efficient algorithms to compute the SV by leveraging the locality of KNN models.

3. VALUING DATA FOR KNN CLASSIFIERS

In this section, we present an algorithm that can calculate the exact SV for KNN classifiers in quasi-linear time. Further, we exhibit an approximate algorithm based on LSH that could achieve sublinear complexity.

3.1 Exact SV Calculation

KNN algorithms are popular supervised learning methods, widely adopted in a multitude of applications such as computer vision, information retrieval, etc. Suppose the dataset D consisting of pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ taking values in $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the feature space and $\mathcal{Y}$ is the label space. Depending on whether the nearest neighbor algorithm is used for classification or regression, $\mathcal{Y}$ is either discrete or continuous. The training phase of KNN consists only of storing the features and labels in D. The testing phase is aimed at finding the label for a given query (or test) feature. This is done by searching for the K training features most similar to the query feature and assigning a label to the query according to the labels of its K nearest neighbors. Given a single testing point x_{test} with the label y_{test}, the simplest, unweighted version of a KNN classifier first finds the top-K training points $(x_{\alpha_1}, \cdots, x_{\alpha_K})$ that are most similar to x_{test} and outputs the probability of x_{test} taking the label y_{test} as $P[x_{\text{test}} \to y_{\text{test}}] = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}[y_{\alpha_k} = y_{\text{test}}]$, where α_k is the index of the kth nearest neighbor.

One natural way to define the utility of a KNN classifier is by the likelihood of the right label:

$$\nu(S) = \frac{1}{K} \sum_{k=1}^{\min\{K,|S|\}} \mathbb{1}[y_{\alpha_k(S)} = y_{\text{test}}] \tag{4}$$

where $\alpha_k(S)$ represents the index of the training feature that is kth closest to x_{test} among the training examples in S. Specifically, $\alpha_k(I)$ is abbreviated to α_k.

Using this utility function, we can derive an efficient, but exact way of computing the SV.

THEOREM 1. *Consider the utility function in (4). Then, the SV of each training point can be calculated recursively as follows:*

$$s_{\alpha_N} = \frac{\mathbb{1}[y_{\alpha_N} = y_{test}]}{N} \tag{5}$$

$$s_{\alpha_i} = s_{\alpha_{i+1}} + \frac{\mathbb{1}[y_{\alpha_i} = y_{test}] - \mathbb{1}[y_{\alpha_{i+1}} = y_{test}]}{K} \frac{\min\{K, i\}}{i} \tag{6}$$

Note that the above result for a single test point can be readily extended to the multiple-test-point case, in which the utility function is defined by

$$\nu(S) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \frac{1}{K} \sum_{k=1}^{\min\{K,|S|\}} \mathbb{1}[y_{\alpha_k^{(j)}(S)} = y_{\text{test},j}] \tag{7}$$

where $\alpha_k^{(j)}(S)$ is the index of the kth nearest neighbor in S to $x_{\text{test},j}$. By the additivity property, the SV for multiple test points is the average of the SV for every single test point. The pseudo-code for calculating the SV for an unweighted KNN classifier is summarized in the appendix. The computational complexity is only $\mathcal{O}(N \log N N_{\text{test}})$ for N training data points and N_{test} test data points—this is simply to sort N_{test} arrays of N numbers!

The proof of Theorem 1 relies on the following lemma, which states that the difference in the utility gain induced by either point i or point j translates linearly to the difference in the respective SVs.

LEMMA 1. *For any $i, j \in I$, the difference in SVs between i and j is*

$$s_i - s_j = \frac{1}{N-1} \sum_{S \subseteq I \setminus \{i,j\}} \frac{\nu(S \cup \{i\}) - \nu(S \cup \{j\})}{\binom{N-2}{|S|}} \tag{8}$$

PROOF OF THEOREM 1. W.l.o.g., we assume that $x_1, \ldots, x_n$ are sorted according to their similarity to x_{test}, that is, $x_i = x_{\alpha_i}$. For any given subset $S \subseteq I \setminus \{i, i+1\}$ of size k, we split the subset into two disjoint sets S_1 and S_2 such that $S = S_1 \cup S_2$ and $|S_1| + |S_2| = |S| = k$. Given two neighboring points with indices $i, i+1 \in I$, we constrain S_1 and S_2 to $S_1 \subseteq \{1, ..., i-1\}$ and $S_2 \subseteq \{i+2, ..., N\}$.

Let s_i be the SV of data point x_i. By Lemma 1, we can draw conclusions about the SV difference $s_i - s_{i+1}$ by inspecting the utility difference $\nu(S \cup \{i\}) - \nu(S \cup \{i+1\})$ for any $S \subseteq I \setminus$

$\{i, i+1\}$. We analyze $\nu(S\cup\{i\})-\nu(S\cup\{i+1\})$ by considering the following cases.

(1) $|S_1| \geq K$**.** In this case, we know that $i, i+1 > K$ and therefore $\nu(S\cup\{i\}) = \nu(S\cup\{i+1\}) = \nu(S)$, hence $\nu(S\cup\{i\})-\nu(S\cup\{i+1\})=0$.

(2) $|S_1| < K$**.** In this case, we know that $i \leq K$ and therefore $\nu(S\cup\{i\})-\nu(S)$ might be nonzero. Note that including a point i into S can only expel the Kth nearest neighbor from the original set of K nearest neighbors. Thus, $\nu(S\cup\{i\})-\nu(S) = \frac{1}{K}(\mathbb{1}[y_i = y_{\text{test}}] - \mathbb{1}[y_K = y_{\text{test}}])$. The same hold for the inclusion of point $i+1$: $\nu(S\cup\{i+1\})-\nu(S) = \frac{1}{K}(\mathbb{1}[y_{i+1} = y_{\text{test}}] - \mathbb{1}[y_K = y_{\text{test}}])$. Combining the two equations, we have

$$\nu(S\cup\{i\})-\nu(S\cup\{i+1\}) = \frac{\mathbb{1}[y_i = y_{\text{test}}] - \mathbb{1}[y_{i+1} = y_{\text{test}}]}{K}$$

Combining the two cases discussed above and applying Lemma 1, we have

$$\begin{aligned}
&s_i - s_{i+1}\\
=&\frac{1}{N-1}\sum_{k=0}^{N-2}\frac{1}{\binom{N-2}{k}}\sum_{\substack{S_1\subseteq\{1,\ldots,i-1\},\\ S_2\subseteq\{i+2,\ldots,N\}:\\ |S_1|+|S_2|=k, |S_1|<K}}\frac{\mathbb{1}[y_i = y_{\text{test}}] - \mathbb{1}[y_{i+1} = y_{\text{test}}]}{K}\\
=&\frac{\mathbb{1}[y_i = y_{\text{test}}] - \mathbb{1}[y_{i+1} = y_{\text{test}}]}{K}\\
\times&\frac{1}{N-1}\sum_{k=0}^{N-2}\frac{1}{\binom{N-2}{k}}\sum_{m=0}^{\min(K-1,k)}\binom{i-1}{m}\binom{N-i-1}{k-m} \qquad (9)
\end{aligned}$$

The sum of binomial coefficients in (9) can be simplified as follows:

$$\begin{aligned}
&\sum_{k=0}^{N-2}\frac{1}{\binom{N-2}{k}}\sum_{m=0}^{\min\{K-1,k\}}\binom{i-1}{m}\binom{N-i-1}{k-m} \qquad (10)\\
=&\sum_{m=0}^{\min\{K-1,i-1\}}\sum_{k'=0}^{N-i-1}\frac{\binom{i-1}{m}\binom{N-i-1}{k'}}{\binom{N-2}{m+k'}} \qquad (11)\\
=&\frac{\min\{K,i\}(N-1)}{i} \qquad (12)
\end{aligned}$$

where the first equality is due to the exchange of the inner and outer summation and the second one is by taking $v = N-i-1$ and $u = i-1$ in the binomial identity $\sum_{j=0}^{v}\frac{\binom{u}{i}\binom{v}{j}}{\binom{u+v}{i+j}} = \frac{u+v+1}{v+1}$.

Therefore, we have the following recursion

$$s_i - s_{i+1} = \frac{\mathbb{1}[y_i = y_{\text{test}}] - \mathbb{1}[y_{i+1} = y_{\text{test}}]}{K}\frac{\min\{K,i\}}{i} \qquad (13)$$

Now, we analyze the formula for s_N, the starting point of the recursion. Since x_N is farthest to x_{test} among all training points, x_N results in non-zero marginal utility only when it is added to the subsets of size smaller than K. Hence, s_N can be written as

$$\begin{aligned}
s_N =& \frac{1}{N}\sum_{k=0}^{K-1}\frac{1}{\binom{N-1}{k}}\sum_{|S|=k, S\subseteq I\setminus\{N\}}\nu(S\cup N)-\nu(S) \qquad (14)\\
=& \frac{1}{N}\sum_{k=0}^{K-1}\frac{1}{\binom{N-1}{k}}\sum_{|S|=k, S\subseteq I\setminus\{N\}}\frac{\mathbb{1}[y_N = y_{\text{test}}]}{K} \qquad (15)\\
=& \frac{\mathbb{1}[y_N = y_{\text{test}}]}{N} \qquad (16)
\end{aligned}$$

□

3.2 LSH-based Approximation

The exact calculation of the KNN SV for a query instance requires to sort the entire training dataset, and has computation complexity $\mathcal{O}(N_{\text{test}}(Nd + N\log(N)))$, where d is the feature dimension. Thus, the exact method becomes expensive for large and high-dimensional datasets. We now present a sublinear algorithm to approximate the KNN SV for classification tasks.

The key to boosting efficiency is to realize that only $\mathcal{O}(1/\epsilon)$ nearest neighbors are needed to estimate the KNN SV with up to ϵ error. Therefore, we can avert the need of sorting the entire database for every new query point.

THEOREM 2. *Consider the utility function defined in (4). Consider $\{\hat{s}_i\}_{i=1}^N$ defined recursively by*

$$\hat{s}_{\alpha_i} = 0 \qquad \text{if } i \geq K^* \qquad (17)$$

$$\hat{s}_{\alpha_i} = \hat{s}_{\alpha_{i+1}} + \frac{\mathbb{1}[y_{\alpha_i} = y_{test}] - \mathbb{1}[y_{\alpha_{i+1}} = y_{test}]}{K}\frac{\min\{K,i\}}{i} \qquad \text{if } i \leq K^*-1 \qquad (18)$$

where $K^* = \max\{K, \lceil 1/\epsilon\rceil\}$ for some $\epsilon > 0$. Then, $[\hat{s}_{\alpha_1},\ldots, \hat{s}_{\alpha_N}]$ is an $(\epsilon, 0)$-approximation to the true SV $[s_{\alpha_1},\ldots, s_{\alpha_N}]$ and $\hat{s}_i - \hat{s}_{i+1} = s_i - s_{i+1}$ for $i \leq K^*-1$.

Theorem 2 indicates that we only need to find $\max\{K, \lceil 1/\epsilon\rceil\}(\triangleq K^*)$ nearest neighbors to obtain an $(\epsilon, 0)$-approximation. Moreover, since $\hat{s}_i - \hat{s}_{i+1} = s_i - s_{i+1}$ for $i \leq K^*-1$, the approximation retains the original value rank for K^* nearest neighbors.

The question on how to efficiently retrieve nearest neighbors to a query in large-scale databases has been studied extensively in the past decade. Various techniques, such as the kd-tree [43], LSH [15], have been proposed to find approximate nearest neighbors. Although all of these techniques can potentially help improve the efficiency of the data valuation algorithms for KNN, we focus on LSH in this paper, as it was experimentally shown to achieve large speedup over several tree-based data structures [15, 22, 23]. In LSH, every training instance x is converted into codes in each hash table by using a series of hash functions $h_j(x)$, $j = 1,\ldots,m$. Each hash function is designed to preserve the relative distance between different training instances; similar instances have the same hashed value with high probability. Various hash functions have been proposed to approximate KNN under different distance metrics [10, 15]. We will focus on the distance measured in l_2 norm; in that case, a commonly used hash function is $h(x) = \left\lfloor\frac{w^T x+b}{r}\right\rfloor$, where w is a vector with entries sampled from a p-stable distribution, and b is uniformly chosen from the range $[0, r]$. It is shown in [15]:

$$P[h(x_i) = h(x_{\text{test}})] = f_h(\|x_i - x_{\text{test}}\|_2) \qquad (19)$$

where the function $f_h(c) = \int_0^r \frac{1}{c}f_2(\frac{z}{c})(1-\frac{z}{r})dz$ is a monotonically decreasing with c. Here, f_2 is the probability density function of the absolute value of a 2-stable random variable.

We now present a theorem which relates the success rate of finding approximate nearest neighbors to the intrinsic property of the dataset and the parameters of LSH.

THEOREM 3. *LSH with $\mathcal{O}(d\log(N)N^{g(C_K)}\log\frac{K}{\delta})$ time complexity, $\mathcal{O}(Nd+N^{g(C_K)+1}\log\frac{K}{\delta})$ space complexity, and $\mathcal{O}(N^{g(C_K)}\log\frac{K}{\delta})$ hash tables can find the exact K nearest neighbors with probability $1-\delta$, where $g(C_K) = \log f_h(1/C_K)/\log f_h(1)$ is a monotonically decreasing function. $C_K = D_{mean}/D_K$, where D_{mean} is the expected distance of a random training instance to a*

query x_{test} and D_K is the expected distance between x_{test} to its Kth nearest neighbor denoted by $x_{\alpha_i}(x_{test})$, i.e.,

$$D_{mean} = \mathbb{E}_{x,x_{test}}[D(x, x_{test})] \tag{20}$$

$$D_K = \mathbb{E}_{x_{test}}[D(x_{\alpha_i}(x_{test}), x_{test}] \tag{21}$$

The above theorem essentially extends the 1NN hardness analysis in Theorem 3.1 of [26] to KNN. C_K measures the ratio between the distance from a query instance to a random training instance and that to its Kth nearest neighbor. We will hereinafter refer to C_K as *Kth relative contrast*. Intuitively, C_K signifies the difficulty of finding the Kth nearest neighbor. A smaller C_K implies that some random training instances are likely to have the same hashed value as the Kth nearest neighbor, thus entailing a high computational cost to differentiate the true nearest neighbors from the false positives. Theorem 3 shows that among the datasets of the same size, the one with higher relative contrast will need lower time and space complexity and fewer hash tables to approximate the K nearest neighbors. Combining Theorem 2 and Theorem 3, we obtain the following theorem that explicates the tradeoff between KNN SV approximation errors and computational complexity.

THEOREM 4. *Consider the utility function defined in (7). Let $\hat{x}_{\alpha_k^{(j)}}$ denote the kth closest training point to $x_{test,j}$ output by LSH with $\mathcal{O}(N_{test}d\log(N)N^{g(C_{K^*})}\log\frac{N_{test}K^*}{\delta})$ time complexity, $\mathcal{O}(Nd+N^{g(C_{K^*})+1}\log\frac{N_{test}K^*}{\delta})$ space complexity, and $\mathcal{O}(N^{g(C_{K^*})}\log\frac{N_{test}K^*}{\delta})$ hash tables, where $K^* = \max(K, \lceil 1/\epsilon \rceil)$. Suppose that $\{\hat{s}_i\}_{i=1}^N$ is computed via $\hat{s}_i = \frac{1}{N_{test}}\sum_{j=1}^{N_{test}}\hat{s}_{i,j}$ and $\hat{s}_{i,j}$ $(j = 1, \ldots, N_{test})$ are defined recursively by*

$$\hat{s}_{\alpha_i^{(j)},j} = 0 \qquad \text{if } i \geq K^* \tag{22}$$

$$\hat{s}_{\alpha_i^{(j)},j} = \hat{s}_{\alpha_{i+1}^{(j)},j} \qquad \text{if } i \leq K^* - 1$$
$$+ \frac{\mathbb{1}[\hat{y}_{\alpha_i^{(j)}} = y_{test,j}] - \mathbb{1}[\hat{y}_{\alpha_{i+1}^{(j)}} = y_{test,j}]}{K}\frac{\min\{K,i\}}{i} \tag{23}$$

where $\hat{y}_{\alpha_i^{(j)}}$ and $y_{test,j}$ are the labels associated with $\hat{x}_{\alpha_i^{(j)}}$ and $x_{test,j}$, respectively. Let the true SV of $\hat{x}_{\alpha_k}$ be denoted by s_{α_i}. Then, $[\hat{s}_{\alpha_1}, \ldots, \hat{s}_{\alpha_N}]$ is an (ϵ, δ)-approximation to the true SV $[s_{\alpha_1}, \ldots, s_{\alpha_N}]$.

The gist of the LSH-based approximation is to focus only on the SV of the retrieved nearest neighbors and neglect the values of the rest of the training points since their values are small enough. For a error requirement ϵ not too small such that $C_{K^*} > 1$, the LSH-based approximation has sublinear time complexity, thus enjoying higher efficiency than the exact algorithm.

4. EXTENSIONS

We extend the exact algorithm for unweighted KNN to other settings. Specifically, as illustrated by Figure 3, we categorize a data valuation problem according to whether data contributors are valued in tandem with a data analyst; whether each data contributor provides a single data instance or multiple ones; whether the underlying ML model is a weighted KNN or unweighted; and whether the model solves a regression or a classification task. We will discuss the valuation algorithm for each of the above settings.

Unweighted KNN Regression. For regression tasks, we define the utility function by the negative mean square error of an unweighted KNN regressor:

$$U(S) = -\left(\frac{1}{K}\sum_{k=1}^{\min\{K,|S|\}} y_{\alpha_k(S)} - y_{\text{test}}\right)^2 \tag{24}$$

Using similar proof techniques to Theorem 1, we provide a simple iterative procedure to compute the SV for unweighted KNN regression in the appendix.

Weighted KNN. A weighted KNN estimate produced by a training set S can be expressed as $\hat{y}(S) = \sum_{k=1}^{\min\{K,|S|\}} w_{\alpha_k(S)} y_{\alpha_k}$, where $w_{\alpha_k(S)}$ is the weight associated with the kth nearest neighbor in S. The weight assigned to a neighbor in the weighted KNN estimate often varies with the neighbor-to-test distance so that the evidence from more nearby neighbors is weighted more heavily [19]. Correspondingly, we define the utility function associated with weighted KNN classification and regression tasks as

$$U(S) = \sum_{k=1}^{\min\{K,|S|\}} w_{\alpha_k(S)}\mathbb{1}[y_{\alpha_k(S)} = y_{\text{test}}] \tag{25}$$

and

$$U(S) = -\left(\sum_{k=1}^{\min\{K,|S|\}} w_{\alpha_k(S)} y_{\alpha_k(S)} - y_{\text{test}}\right)^2. \tag{26}$$

For weighted KNN classification and regression, the SV can no longer be computed exactly in $\mathcal{O}(N\log N)$ time. In the appendix, we present a theorem showing that it is however possible to compute the exact SV for weighted KNN in $\mathcal{O}(N^K)$ time. The right figure illustrates the origin of the polynomial complexity result. When applying (2) to KNN, we only need to focus on the subsets whose utility might be affected by the addition of ith training instance. Since there are only N^K possible distinctive combinations for K nearest neighbors, the number of distinct utility values for all $S \subseteq I$ is upper bounded by N^K.

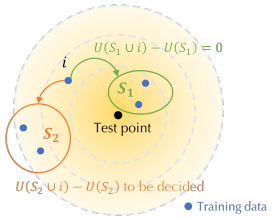

Figure 4: Illustration of the idea to compute the SV for weighted KNN.

Multiple Data Per Contributor. We now study the case where each seller provides more than one data instance. The goal is to fairly value individual sellers in lieu of individual training points. In the appendix, we show that for both unweighted/weighted classifiers/regressors, the complexity for computing the SV of each seller is $\mathcal{O}(M^K)$, where M is the number of sellers. Particularly, when $K = 1$, even though each seller can provision multiple instances, the utility function only depends on the training point that is nearest to the query point. Thus, for 1NN, the problem of computing the multi-data-per-seller KNN SV reduces to the single-data-per-seller case; thus, the corresponding computational complexity is $\mathcal{O}(M\log M)$.

Valuing Computation. Oftentimes, the buyer may outsource data analytics to a third party, which we call the analyst throughout the rest of the paper. The analyst analyzes the training dataset aggregated from different sellers and returns an ML model to the buyer. In this process, the analyst contributes various computation efforts, which may include intellectual property pertaining to data

anlytics, usage of computing infrastructure, among others. Here, we want to address the problem of appraising both sellers (data contributors) and analysts (computation contributors) within a unified game-theoretic framework.

Firstly, we extend the game-theoretic framework for data valuation to model the interplay between data and computation. The resultant game is termed a *composite game*. By contrast, the game discussed previously which involves only the sellers is termed a *data-only game*. In the composite game, there are $M+1$ players, consisting of M sellers denoted by I^s and one analyst denoted by C. We can express the utility function ν_c associated with the game in terms of the utility function ν in the data-only game as follows. Since in the case of outsourced analytics, both contributions from data sellers and data analysts are necessary for building models, the value of a set $S \subseteq I^s \cup \{C\}$ in the composite game is zero if S only contains the sellers or the analyst; otherwise, it is equal to ν evaluated on all the sellers in S. Formally, we define the utility function ν_c by

$$\nu_c(S) = \begin{cases} 0, \text{ if } S = \{C\} \text{ or } S \subseteq I^s \\ \nu(S \setminus \{C\}), \text{ otherwise} \end{cases} \tag{27}$$

The goal in the composite game is to allocate $\nu_c(\{I^s, C\})$ to the individual sellers and the analyst. $s(\nu_c, i)$ and $s(\nu_c, C)$ represent the value received by seller i and the analyst, respectively. We suppress the dependency of s on the utility function whenever it is self-evident, denoting the value allocated to seller i and the analyst by s_i and s_c, respectively.

In the appendix, we show that one can compute the SV for both the sellers and the analyst with the same computational complexity as the one needed for the data-only game.

Comments on the Proof Techniques. We have shown that we can circumvent the exponential complexity for computing the SV for a standard unweighted KNN classifier and its extensions. A natural question is whether it is possible to abstract the commonality of these cases and provide a general property of the utility function that one can exploit to derive efficient algorithms.

Suppose that some group of S's induce the same $\nu(S \cup \{i\}) - \nu(S \cup \{j\})$ and there only exists T number of such groups. More formally, consider that $\nu(S \cup \{i\}) - \nu(S \cup \{j\})$ can be represented by a "piecewise" form:

$$\nu(S \cup \{i\}) - \nu(S \cup \{j\}) = \sum_{t=1}^{T} C_{ij}^{(t)} \mathbb{1}[S \in \mathcal{S}_t] \tag{28}$$

where $\mathcal{S}_t \subseteq 2^{I \setminus \{i,j\}}$ and $C_{i,j}^{(t)} \in \mathbb{R}$ is a constant associated with tth "group." An application of Lemma 1 to the utility functions with the piecewise utility difference form indicates that the SV difference between i and j is

$$s_i - s_j = \frac{1}{N-1} \sum_{S \subseteq I \setminus \{i,j\}} \sum_{t=1}^{T} \frac{C_{ij}^{(t)}}{\binom{N-2}{|S|}} \mathbb{1}[S \in \mathcal{S}_t] \tag{29}$$

$$= \frac{1}{N-1} \sum_{t=1}^{T} C_{ij}^{(t)} \left[\sum_{k=0}^{N-2} \frac{|\{S : S \in \mathcal{S}_t, |S| = k\}|}{\binom{N-2}{k}} \right] \tag{30}$$

With the piecewise property (28), the SV calculation is reduced to a counting problem. As long as the quantity in the bracket of (30) can be efficiently evaluated, the SV difference between any pair of training points can be computed in $\mathcal{O}(TN)$.

Indeed, one can verify that the utility function for unweighted KNN classification, regression and weighted KNN have the aforementioned "piecewise" utility difference property with $T = 1, N-1, \sum_{k=0}^{K} \binom{N-2}{k}$, respectively. More details can be found in the appendix.

5. IMPROVED MC APPROXIMATION

As discussed previously, the SV for unweighted KNN classification and regression can be computed exactly with $\mathcal{O}(N \log N)$ complexity. However, for the variants including the weighted KNN and multiple-data-per-seller KNN, the complexity to compute the exact SV is $\mathcal{O}(N^K)$ and $\mathcal{O}(M^K)$, respectively, which are clearly not scalable. We propose a more efficient way to evaluate the SV up to provable approximation errors, which modifies the existing MC algorithm presented in Section 2.2. By exploiting the locality property of the KNN-type algorithms, we propose a tighter upper bound on the number of permutations for a given approximation error and exhibit a novel implementation of the algorithm using efficient data structures.

The existing sample complexity bound is based on Hoeffding's inequality, which bounds the number of permutations needed in terms of the range of utility difference ϕ_i. This bound is not always optimal as it depends on the extremal values that a random variable can take and thus accounts for the worst case. For KNN, the utility does not change after adding training instance i for many subsets; therefore, the variance of ϕ_i is much smaller than its range. This inspires us to use Bennett's inequality, which bounds the sample complexity in terms of the variance of a random variable and often results in a much tighter bound than Hoeffding's inequality.

THEOREM 5. *Given the range $[-r, r]$ of the utility difference ϕ_i, an error bound ϵ, and a confidence $1-\delta$, the sample size required such that*

$$P[\|\hat{s} - s\|_\infty \geq \epsilon] \leq \delta$$

is $T \geq T^*$. T^* is the solution of

$$\sum_{i=1}^{N} \exp(-T^*(1-q_i^2) h(\frac{\epsilon}{(1-q_i^2)r})) = \delta/2. \tag{31}$$

where $h(u) = (1+u)\log(1+u) - u$ and

$$q_i = \begin{cases} 0, \ i = 1, \ldots, K \\ \frac{i-K}{i}, \ i = K+1, \ldots, N \end{cases} \tag{32}$$

Given ϵ, δ, and r, the required permutation size T^* derived from Bennett's bound can be computed numerically. For general utility functions the range r of the utility difference is twice the range of the utility function, while for the special case of the unweighted KNN classifier, $r = \frac{1}{K}$.

Although determining exact T^* requires numerical calculation, we can nevertheless gain insights into the relationship between N, ϵ, δ and T^* through some approximation. We leave the detailed derivation to the appendix, but it is often reasonable to use the following $\tilde{T}$ as an approximation of T^*:

$$\tilde{T} \geq \frac{r^2}{\epsilon^2} \log \frac{2K}{\delta} \tag{33}$$

The sample complexity bound derived above does not change with N. On the one hand, a larger training data size implies more unknown SVs to be estimated, thus requiring more random permutations. On the other hand, the variance of the SV across all training data decreases with the training data size, because an increasing proportion of training points makes insignificant contributions to the query result and results in small SVs. These two opposite driving forces make the required permutation size about the same across all training data sizes.

Algorithm 1: Improved MC Approach

input : Training set - $D = \{(x_i, y_i)\}_{i=1}^{N}$, utility function $\nu(\cdot)$, the number of measurements - M, the number of permutations - T
output : The SV of each training point - $\hat{s} \in \mathbb{R}^N$

for $t \leftarrow 1$ **to** T **do**
  $\pi_t \leftarrow$ GenerateUniformRandomPermutation(D);
  Initialize a length-K max-heap H to maintain the KNN;
  for $i \leftarrow 1$ **to** N **do**
    Insert $\pi_{t,i}$ to H;
    if H *changes* **then**
      $\phi^t_{\pi_{t,i}} \leftarrow \nu(\pi_{t,1:i}) - \nu(\pi_{t,1:i-1})$;
    else
      $\phi^t_{\pi_{t,i}} \leftarrow \phi^t_{\pi_{t,i-1}}$;
    end
  end
end
$\hat{s}_i = \frac{1}{T}\sum_{t=1}^{T} \phi_i^t$ for $i = 1, \ldots, N$;

The algorithm for the improved MC approximation is provided in Algorithm 1. We use a max-heap to organize the KNN. Since inserting any training data to the heap costs $\mathcal{O}(\log K)$, incrementally updating the KNN in a permutation costs $\mathcal{O}(N \log K)$. Using the bound on the number of permutations in (33), we can show that the total time complexity for our improved MC algorithm is $\mathcal{O}(\frac{N}{\epsilon^2} \log K \log \frac{K}{\delta})$.

6. EXPERIMENTS

We evaluate the proposed approaches to computing the SV of training data for various nearest neighbor algorithms.

6.1 Experimental Setup

Datasets. We used the following popular benchmark datasets of different sizes: (1) `dog-fish` [31] contains the features of dog and cat images extracted from ImageNet, with 900 training examples and 300 test examples for each class. The features have 2048 dimensions, generated by the state-of-the-art Inception v3 network [48] with all but the top layer. (2) `MNIST` [35] is a handwritten digit dataset with 60000 training images and 10000 test images. We extracted 1024-dimensional features via a convolutional network. (3) The `CIFAR-10` dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. The deep features have 2048 dimensions and were extracted via the ResNet-50 [27]. (4) `ImageNet` [17] is an image dataset with more than 1 million images organized according to the WordNet hierarchy. We chose 1000 classes which have in total around 1 million images and extracted 2048-dimensional deep features by the ResNet-50 network. (5) Yahoo Flickr Creative Commons 100M that consists of 99.2 million photos. We randomly chose a 10-million subset (referred to as `Yahoo10m` hereinafter) for our experiment, and used the deep features extracted by [7].

Parameter selection for LSH. The three main parameters that affect the performance of the LSH are the number of projections per hash value (m), the number of hash tables (h), and the width of the project (r). Decreasing r decreases the probability of collision for any two points, which is equivalent to increasing m. Since a smaller m will lead to better efficiency, we would like to set r as small as possible. However, decreasing r below a certain threshold increases the quantity $g(C_K)$, thereby requiring us to increase h. Following [15], we performed grid search to find the optimal value of r which we used in our experiments. Following [22], we set $m = \alpha \log N / \log(f_h(D_{\text{mean}})^{-1})$. For a given value of m, it is easy to find the optimal value of h which will guarantee that the SV approximation error is no more than a user-specified threshold.

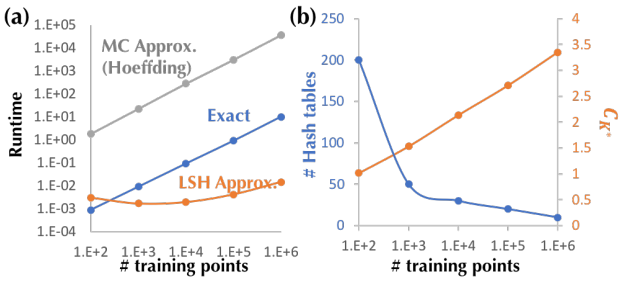

Figure 6: Performance of unweighted KNN classification in the single-data-per-seller case.

We tried a few values for α and reported the m that leads to lowest runtime. For all experiments pertaining to the LSH, we divided the dataset into two disjoint parts: one for selecting the parameters, and another for testing the performance of LSH for computing the SV.

6.2 Experimental Results

6.2.1 *Unweighted KNN Classifier*

Correctness. We first empirically validate our theortical result. We randomly selected 1000 training points and 100 test points from `MNIST`. We computed the SV of each training point with respect to the KNN utility using the exact algorithm and the baseline MC method. We see that the MC estimate of the SV for each training point converges to the result of the exact algorithm.

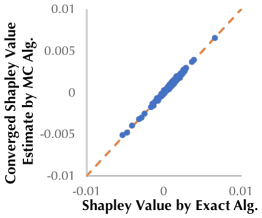

Figure 5: The SV produced by the exact algorithm and the baseline MC approximation algorithm.

Performance. We validated the hypothesis that our exact algorithm and the LSH-based method outperform the baseline MC method. We take the approximation error $\epsilon = 0.1$ and $\delta = 0.1$ for both MC and LSH-based approximations. We bootstrapped the `MNIST` dataset to synthesize training datasets of various sizes. The three SV calculation methods were implemented on a machine with 2.6 GHz Intel Core i7 CPU. The runtime of the three methods for different datasets is illustrated in Figure 6 (a). The proposed exact algorithm is faster than the baseline approximation by several orders magnitude and it produces the exact SV. By circumventing the computational complexity of sorting a large array, the LSH-based approximation can significantly outperform the exact algorithm, especially when the training size is large. Figure 6 (b) sheds light on the increasing performance gap between the LSH-based approximation and the exact method with respect to the training size. The relative contrast of these bootstrapped datasets grows with the number of training points, thus requiring fewer hash tables and less time to search for approximate nearest neighbors. We also tested the approximation approach proposed in our prior work [29], which achieves the-start-of-the-art performance for ML models that cannot be incrementally maintained. However, for models that have efficient incremental training algorithms, like KNN, it is less efficient than the baseline approximation, and the experiment for 1000 training points did not finish in 4 hours.

Using a machine with the Intel Xeon E5-2690 CPU and 256 GB RAM, we benchmarked the runtime of the exact and the LSH-

Figure 7: Average runtime of the exact and the LSH-based approximation algorithm for computing the unweighted KNN SV for a single test point. We take $\epsilon, \delta = 0.1$ and $K = 1$.

Dataset	Size	Estimated Contrast	Runtime (Exact)	Runtime (LSH)
CIFAR-10	6E+4	1.2802	0.78s	0.23s
ImageNet	1E+6	1.2163	11.34s	2.74s
Yahoo10m	1E+7	1.3456	203.43s	44.13s

Figure 8: Comparison of prediction accuracy of KNN vs. logistic regression on deep features.

Dataset	1NN	2NN	5NN	Logistic Regression
CIFAR-10	81%	83%	80%	**87%**
ImageNet	77%	73%	**84%**	82%
Yahoo10m	90%	96%	**98%**	96%

based approximation algorithm on three popular datasets, including CIFAR-10, ImageNet, and Yahoo10m. For each dataset, we randomly selected 100 test points, computed the SV of all training points with respect to each test point, and reported the average runtime across all test points. The results for $K = 1$ are reported in Figure 7. We can see that the LSH-based method can bring a $3\times$-$5\times$ speed-up compared with the exact algorithm. The performance of LSH depends heavily on the dataset, especially in terms of its relative contrast. This effect will be thoroughly studied in the sequel. We compare the prediction accuracy of KNN ($K = 1, 2, 5$) with the commonly used logistic regression and the result is illustrated in Figure 8. We can see that KNN achieves comparable prediction power to logistic regression when using features extracted via deep neural networks. The runtime of the exact and the LSH-based approximation for $K = 2, 5$ is similar to the $K = 1$ case in Figure 7, so we will leave their corresponding results to the arXiv version.

Effect of relative contrast on the LSH-based method. Our theoretical result suggests that the K^*th relative contrast ($K^* = \max\{K, \lceil 1/\epsilon \rceil\}$) determines the complexity of the LSH-based approximation. We verified the effect of relative contrast by experimenting on three datasets, namely, dog-fish, deep and gist. deep and gist were constructed by extracting the deep features and gist features [47] from MNIST, respectively. All of these datasets were normalized such that $D_{\text{mean}} = 1$. Figure 9 (a) shows that the relative contrast of each dataset decreases as K^* increases. In this experiment, we take $\epsilon = 0.01$ and $K = 2$, so the corresponding $K^* = 1/\epsilon = 100$. At this value of K^*, the relative contrast is in the following order: deep (1.57) > gist (1.48) > dog-fish (1.17). From Figure 9 (b) and (c), we see that the number of hash tables and the number of returned points required to meet the ϵ error tolerance for the three datasets follow the reversed order of their relative contrast, as predicted by Theorem 4. Therefore, the LSH-based approximation will be less efficient if the K in the nearest neighbor algorithm is very large or the desired error ϵ is small. Figure 9 (d) shows that the LSH-based method can better approximate the true SV as the recall of the underlying nearest neighbor retrieval gets higher. For the datasets with high relative contrast, e.g., deep and gist, a moderate value of recall (~ 0.7) can already lead to an approximation error below the desired threshold. On the other hand, dog-fish, which has low relative contrast, will need fairly accurate nearest neighbor retrieval (recall ~ 1) to obtain a tolerable approximation error. The reason for the different retrieval accuracy requirements is that for the dataset with higher relative contrast, even if the retrieval of the nearest neighbors is inaccurate, the rank of the erroneous elements in the retrieved set may still be close to

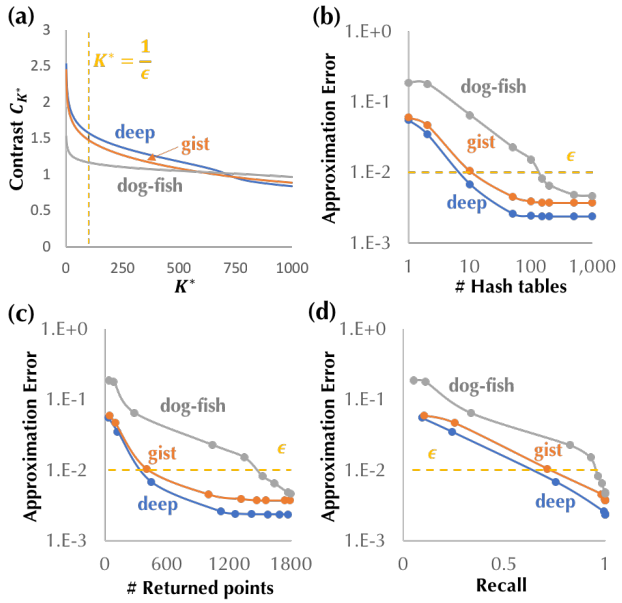

Figure 9: Performance of LSH on three datasets: deep, gist, dog-fish. (a) Relative contrast C_{K^*} vs. K^*. (b), (c) and (d) illustrate the trend of the SV approximation error for different number of hash tables, returned points and recalls.

that of the missed true nearest neighbors. Thus, these erroneous elements will have only little impacts on SV approximation errors.

Simulation of the theoretical bound of LSH. According to Theorem 4, the complexity of the LSH-based approximation is dominated by the exponent $g(C_{K^*})$, where $K^* = \min\{K, 1/\epsilon\}$ and $g(\cdot)$ depends on the width r of the p-stable distribution used for LSH. We computed C_{K^*} and $g(C_{K^*})$ for $\epsilon \in \{0.001, 0.01, 0.1, 1\}$ and let $K = 1$ in this simulation. The orange line in Figure 10 (a) shows that a larger ϵ induces a larger value of relative contrast C_{K^*}, rendering the underlying nearest neighbor retrieval problem of the LSH-based approximation method easier. In particular, C_{K^*} is greater than 1 for all epsilons considered except for $\epsilon = 0.001$. Recall that $g(C_K) = \log f_h(1/C_K)/\log f_h(1)$; thus, $g(C_{K^*})$ will exhibit different trends for the epsilons with $C_{K^*} > 1$ and the ones with $C_{K^*} < 1$, as shown in Figure 10 (b). Moreover, Figure 10 (b) shows that the value of $g(C_{K^*})$ is more or less insensitive to r after a certain point. For ϵ that is not too small, we can choose r to be the value at which $g(C_{K^*})$ is minimized. It does not make sense to use the LSH-based approximation if the desired error ϵ is too small to have the corresponding $g(C_{K^*})$ less than one, since its complexity is theoretically higher than the exact algorithm. The blue line in Figure 10 (a) illustrates the exponent $g(C_{K^*})$ as a function of ϵ when r is chosen to minimize $g(C_{K^*})$. We observe that $g(C_{K^*})$ is always below 1 except when $\epsilon = 0.001$.

6.2.2 *Evaluation of Other Extensions*

We introduced the extensions of the exact SV calculation algorithm to the settings beyond unweighted KNN classification. Some of these settings require polynomial time to compute the exact SV, which is impractical for large-scale datasets. For those settings, we need to resort to the MC approximation method. We first compare the sample complexity of different MC methods, including the baseline and our improved MC method (Section 5). Then, we demonstrate data values computed in various settings.

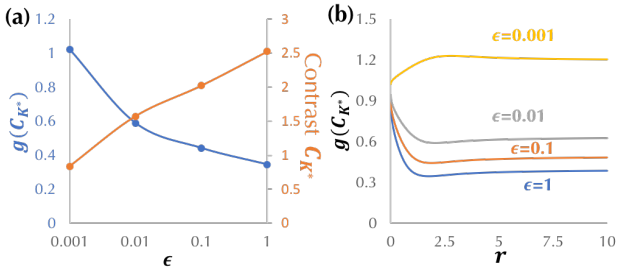

Figure 10: (a) The exponent $g(C_{K^*})$ in the complexity bound of the LSH-based method and the relative contrast C_{K^*} computed for different ϵ. K is fixed to 1. (b) $g(C_{K^*})$ vs. the projection width r of the LSH.

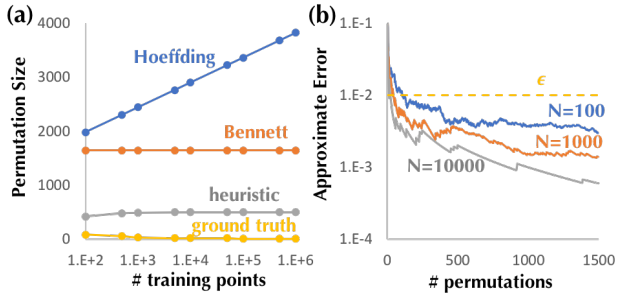

Figure 11: Comparison of the required permutation sizes for different number of training points derived from the Hoeffding's inequality (baseline), Bennett's inequality and the heuristic method against the ground truth.

Sample complexity for MC methods. The time complexity of the MC-based SV approximation algorithms is largely dependent on the number of permutations. Figure 11 compares the permutation sizes used in the following three methods against the actual permutation size needed to achieve a given approximation error (marked as "ground truth" in the figure): (1) "Hoeffding", which is the baseline approach and uses the Hoeffding's inequality to decide the number of permutations; (2) "Bennett", which is our proposed approach and exploits Bennett's inequality to derive the permutation size; (3) "Heuristic", which terminates MC simulations when the change of the SV estimates in the two consecutive iterations is below a certain value, which we set to $\epsilon/50$ in this experiment. We notice that the ground truth requirement for the permutation size decreases at first and remains constant when the training data size is large enough. From Figure 11, the bound based on the Hoeffding's inequality is too loose to correctly predict the correct trend of the required permutation size. By contrast, our bound based on Bennett's inequality exhibits the correct trend of permutation size with respect to training data size. In terms of runtime, our improved MC method based on Bennett's inequality is more than $2\times$ faster than the baseline method when the training size is above 1 million. Moreover, using the aforementioned heuristic, we were able to terminate the MC approximation algorithm even earlier while satisfying the requirement of the approximation error.

Performance. We conducted experiments on the `dog-fish` dataset to compare the runtime of the exact algorithm and our improved MC method. We took $\epsilon = 0.01$ and $\delta = 0.01$ in the approximation algorithm and used the heuristic to decide the stopping iteration.

Figure 12 compares the runtime of the exact algorithm and our improved MC approximation for weighted KNN classification. In the

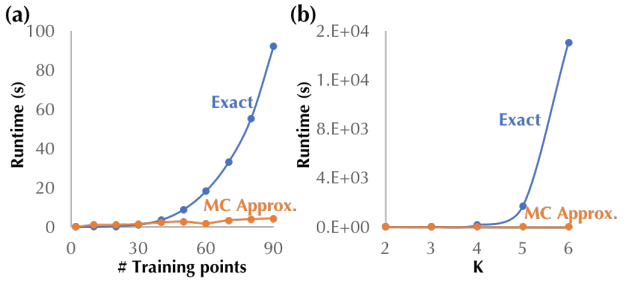

Figure 12: Performance of the weighted KNN classification.

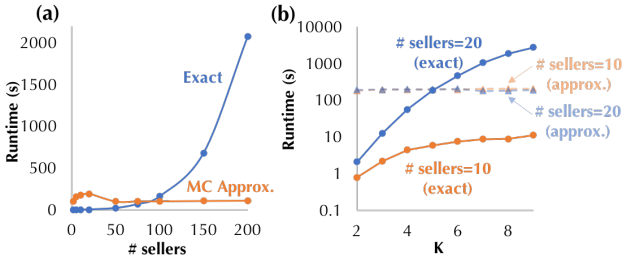

Figure 13: Performance of the KNN classification in the multi-data-per-seller case.

first plot, we fixed $K = 3$ and varied the number of training points. In the second plot, we set the training size to be 100 and changed K. We can see that the runtime of the exact algorithm exhibits polynomial and exponential growth with respect to the training size and K, respectively. By contrast, the runtime of the approximation algorithm increases slightly with the number of training points and remains unchanged for different values of K.

Figure 13 compares the runtime of the exact algorithm and the MC approximation for the unweighted KNN classification when each seller can own multiple data instances. To generate Figure 13 (a), we set $K = 2$ and varied the number of sellers. We kept the total number of training instances of all sellers constant and randomly assigned the same number of training instances to each seller. We can see that the exact calculation of the SV in the multi-data-per-seller case has polynomial time complexity, while the runtime of the approximation algorithm barely changes with the number of sellers. Since the training data in our approximation algorithm were sequentially inserted into a heap, the complexity of the approximation algorithm is mainly determined by the total number of training data held by all sellers. Moreover, as we kept the total number of training points constant, the approximation algorithm appears invariant over the number of sellers. Figure 13 (b) shows that the runtime of exact algorithm increases with K, while the approximation algorithm's runtime is not sensitive to K. To summarize, the approximation algorithm is preferable to the exact algorithm when the number of sellers and K are large.

Unweighted vs. weighted KNN SV. We constructed an unweighted KNN classifier using the `dog-fish`. Figure 14 (a) illustrates the training points with top KNN SVs with respect to a specific test image. We see that the returned images are semantically correlated with the test one. We further trained a weighted KNN on the same training set using the weight function that weighs each nearest neighbor inversely proportional to the distance to a given test point; and compared the SV with the ones obtained from the unweighted KNN classifier. We computed the average SV across all test images for each training point and demonstrated the result in Figure 14 (b). Every point in the figure represents the

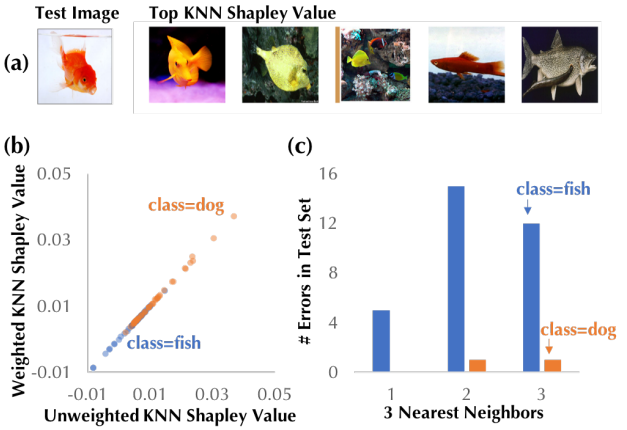

Figure 14: Data valuation on DOG-FISH dataset ($K = 3$). (a) top valued data points; (b) unweighted vs. weighted KNN SV on the whole test set; (c) Per-class top-K neighbors labeled inconsistently with the misclassified test example.

SVs of a training point under the two classifiers. We can see that the unweighted KNN SV is close to the weighted one. This is because in the high-dimensional feature space, the distances from the retrieved nearest neighbors to the query point are large, in which case the weights tend to be small and uniform.

Another observation from Figure 14 (b) is that the KNN SV assigns more values to dog images than fish images. Figure 14 (c) plots the distribution of the number test examples with regard to the number of their top-K neighbors in the training set are with a label inconsistent with the true label of the test example. We see that most of the nearest neighbors with inconsistent labels belong to the fish class. In other words, the fish training images are more close to the dog images in the test set than the dog training images to the test fish. Thus, the fish training images are more susceptible to mislead the predictions and should have lower values. This intuitively explains why the KNN SV places a higher importance on the dog images.

Data-only vs. composite game. We leave the experimental evaluation of the composition game setup to the appendix.

Remarks. We summarize several takeaways from our experimental evaluation. (1) For unweighted KNN classifiers, the LSH-based approximation is more preferable than the exact algorithm when a moderate amount of approximation error can be tolerated and K is relatively small. Otherwise, it is recommended to use the exact algorithm as a default approach for data valuation. (2) For weighted KNN regressors or classifiers, computing the exact SV has $\mathcal{O}(N^K)$ compleixty, thus not scalable for large datasets and large K. Hence, it is recommended to adopt the Monte Carlo method in Algorithm 1. Moreover, using the heuristic based on the change of SV estimates in two consecutive iterations to decide the termination point of the algorithm is much more efficient than using the theoretical bounds, such as Hoeffding or Bennett.

7. DISCUSSION

From the KNN SV to Monetary Reward. Thus far, we have focused on the problem of attributing the KNN utility and its extensions to each data and computation contributor. In practice, the buyer pays a certain amount of money depending on the model utility and it is required to determine the share of each contributor in terms of monetary rewards. Thus, a remaining question is how to map the KNN SV, a share of the total model utility, to a share of the total revenue acquired from the buyer. A simple method for such mapping is to assume that the revenue is an affine function of the model utility, i.e., $R(S) = a\nu(S) + b$ where a and b are some constants which can be determined via market research. Due to the additivity property, we have $s(R, i) = as(\nu, i) + b$. Thus, we can apply the same affine function to the KNN SV to obtain the the monetary reward for each contributor.

Computing the SV for Models Beyond KNN. The efficient algorithms presented in this paper are possible only because of the "locality" property of KNN. However, given many previous empirical results showing that a KNN classifier can often achieve a classification accuracy that is comparable with classifiers such as SVMs and logistic regression given sufficient memory, we could use the KNN SV as a proxy for other classifiers. We compute the SV for a logistic regression classifier and a KNN classifier trained on the same dataset namely `Iris`, and the result shows that the SVs under these two classifiers are indeed correlated (see the above figure). The only caveat is that KNN SV does not distinguish between neighboring data points that have the same label. If this caveat is acceptable, we believe that the KNN SV provides an efficient way to approximately assess the relative contribution of different data points for other classifiers as well. Moreover, for calculating the SV for general deep neural networks, we can take the deep features (i.e., the input to the last softmax layer) and corresponding labels, and train a KNN classifier on the deep features. We calibrate K such that the resulting KNN mimics the performance of the original deep net and then employ the techniques presented in this paper to calculate a surrogate for the SV under the deep net.

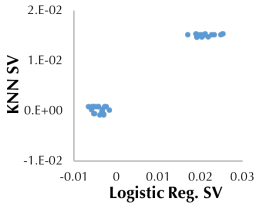

Figure 15: Comparison of the SV for a logistic regression and a KNN trained on the `Iris` dataset.

Implications of Task-Specific Data Valuation. Since the SV depends on the utility function associated with the game, data dividends based on the SV are contingent on the definition of model usefulness in specific ML tasks. The task-specific nature of our data valuation framework offers clear advantages—it allows to accommodate the variability of a data point's utility from one application to another and assess its worth accordingly. Moreover, it enables the data buyer to defend against *data poisoning attacks*, wherein the attacker intentionally contributes adversarial training data points crafted specifically to degrade the performance of the ML model. In our framework, the "bad" training points will naturally have low SVs because they contribute little to boosting the performance of the model.

Having the data values dependent on the ML task, on the other hand, may raise some concerns about whether the data values may inherit the flaws of the ML models as to which the values are computed: if the ML model is biased towards a subpopulation with specific sensitive attributes (e.g., gender, race), will the data values reflect the same bias? Indeed, these concerns can be addressed by designing proper utility functions that devalue the unwanted properties of ML models. For instance, even if the ML model may be biased towards specific subpopulation, the buyer and data contributors can agree on a utility function that gives lower score to unfair models and compute the data values with respect to the concordant utility function. In this case, the training points will be appraised partially according to how much they contribute to improving the model fairness and the resulting data values would not be affected by the bias of the

underlying model. Moreover, there is a venerable line of works studying algorithms to help improve fairness [24, 51, 52]. These algorithms can also be applied to resolve the potential bias in value assignments. For instance, before providing the data to the data buyer, data contributors can preprocess the training data so that the "sanitized" data removes the information correlated with sensitive attributes [52]. However, to ensure that the data values are accurately computed according to an appropriate utility function that the buyer and the data contributors agree on or that the models are trained with proper fairness criteria, it is necessary to develop systems that can support transparent machine learning processes. Recent work has been studying training machine learning models on blockchains for removing the middleman to audit the model performance and enhancing transparency [1]. We are currently implementing the data valuation framework on a blockchain-based data market, which can naturally resolve the problems of transparency and trust. Since the focus of this work is the algorithmic foundation of data valuation, we will leave the discussion of the combination of blockchains and data valuation for future work.

8. RELATED WORK

The problem of data pricing has received a lot of attention recently. The pricing schemes deployed in the existing data marketplaces are simplistic, typically setting a fixed price for the whole or parts of the dataset. Before withdrawn by Microsoft, the Azure Data Marketplace adopted a subscription model that gave users access to a certain number of result pages per month [34]. Xignite [5] sells financial datasets and prices data based on the data type, size, query frequency, etc.

There is rich literature on query-based pricing [16, 32–34, 37, 39, 50], aimed at design pricing schemes for fine-grained queries over a dataset. In query-based pricing, a seller can assign prices to a few views and the price for any queries purchased by a buyer is automatically derived from the explicit prices over the views. Koutris et al. [34] identified two important properties that the pricing function must satisfy, namely, arbitrage-freeness and discount-freeness. The arbitrage-freeness indicates that whenever query Q_1 discloses more information than query Q_2, we want to ensure that the price of Q_1 is higher than Q_2; otherwise, the data buyer has an arbitrage opportunity to purchase the desired information at a lower price. The discount-freeness requires that the prices offer no additional discounts than the ones specified by the data seller. The authors further proved the uniqueness of the pricing function with the two properties, and established a dichotomy on the complexity of the query pricing problem when all views are selection queries. Li et al. [37] proposed additional criteria for data pricing, including non-disclosiveness (preventing the buyers from inferring unpaid query answers by analyzing the publicly available prices of queries) and regret-freeness (ensuring that the price of asking a sequence of queries in multiple interactions is not higher than asking them all-at-once), and investigated the class of pricing functions that meet these criteria. Zheng et al. [53] studied how data uncertainty should affect the price of data, and proposed a data pricing framework for mobile crowd-sensed data. Recent work on query-based pricing focuses on enabling efficient pricing over a wider range of queries, overcoming the issues such as double-charging arising from building practical data marketplaces [16, 33, 50], and compensating data owners for their privacy loss [36]. Due to the increasing pervasiveness of ML-based analytics, there is an emerging interest in studying the cost of acquiring data for ML. Chen et al. [11, 12] proposed a formal framework to price ML model instances, wherein an optimization problem was formulated to find the arbitrage-free price that maximizes the revenue of a seller. The model price can be also used for pricing its training dataset. This paper is complementary to these works in that we consider the scenario where the training set is contributed by multiple sellers and focus on the revenue sharing problem thereof.

While the interaction between data analytics and economics has been extensively studied in the context of both relational database queries and ML, few works have dived into the vital problem of allocating revenues among data owners. [32] presented a technique for fair revenue sharing when multiple sellers are involved in a relational query. By contrast, our paper focuses on the revenue allocation for nearest neighbor algorithms, which are widely adopted in the ML community. Moreover, our approach establishes a formal notion of fairness based on the SV. The use of the SV for pricing personal data can be traced back to [30], which studied the SV in the context of marketing survey, collaborative filtering, and recommendation systems. [13] also applied the SV to quantify the value of personal information when the population of data contributors can be modeled as a network. [42] showed that for specific network games, the exact SV can be computed efficiently.

There exist various methods to rank the importance of training data, which can also potentially be used for data valuation. For instance, influence functions [31] approximate the change of the model performance after removing a training point for smooth parametric ML models. Ogawa et al. [44] proposed rules to identify and remove the least influential data when training support vector machines (SVM) to reduce the computation cost. However, unlike the SV, these approaches do not satisfy the group rationality, fairness, and additivity properties simultaneously.

Despite the desirable properties of the SV, computing the SV is known to be expensive. In its most general form, the SV can be #P-complete to compute [18]. For bounded utility functions, Maleki et al. [41] described a sampling-based approach that requires $\mathcal{O}(N \log N)$ samples to achieve a desired approximation error. By taking into account special properties of the utility function, one can derive more efficient approximation algorithms. For instance, Fatima et al. [20] proposed a probabilistic approximation algorithm with $\mathcal{O}(N)$ complexity for weighted voting games. Ghorbani et al. [21] developed two heuristics to accelerate the estimation of the SV for complex learning algorithms, such as neural networks. One is to truncate the calculation of the marginal contributions as the change in performance by adding only one more training point becomes smaller and smaller. Another is to use one-step gradient to approximate the marginal contribution. The authors also demonstrate the use of the approixmate SV for outlier identification and informed acquisition of new training data. However, their algorithms do not provide any guarantees on the approximation error, thus limiting its viability for practical data valuation. Raskar et al [45] presented a taxonomy of data valuation problems for data markets and discussed challenges associated with data sharing.

9. CONCLUSION

The SV has been long advocated as a useful economic concept to measure data value but has not found its way into practice due to the issue of exponential computational complexity. This paper presents a step towards practical algorithms for data valuation based on the SV. We focus on the case where data are used for training a KNN classifier and develop algorithms that can calculate data values exactly in quasi-linear time and approximate them in sublinear time. We extend the algorithms to the case of KNN regression, the situations where a contributor can own multiple data points, and the task of valuing data contributions and analytics simultaneously. For future work, we will integrate our proposed data valuation algorithms into the clinical data market that we are currently building. We will also explore efficient algorithms to compute the data values for other popular ML algorithms such as gradient boosting, logistic regression, and deep neural networks.

10. REFERENCES

[1] A Decentralized Kaggle: Inside Algorithmias Approach to Blockchain-Based AI Competitions. https://towardsdatascience.com/a-decentralized-kaggle-inside-algorithmias-approach-to-blockchain-based-ai-competitions-8c6aec99e89b.

[2] Dawex. https://www.dawex.com/en/.

[3] Google bigquery. https://cloud.google.com/bigquery/.

[4] Iota. https://data.iota.org/#/.

[5] Xignite. https://apollomapping.com/.

[6] D. Adeniyi, Z. Wei, and Y. Yongquan. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 12(1):90–108, 2016.

[7] G. Amato, F. Falchi, C. Gennaro, and F. Rabitti. Yfcc100m-hnfc6: a large-scale deep features benchmark for similarity search. In *International Conference on Similarity Search and Applications*, pages 196–209. Springer, 2016.

[8] J. J. Bartholdi and E. Kemahlioğlu-Ziya. Using shapley value to allocate savings in a supply chain. In *Supply chain optimization*, pages 169–208. Springer, 2005.

[9] J. Bremer and M. Sonnenschein. Estimating shapley values for fair profit distribution in power planning smart grid coalitions. In *German Conference on Multiagent System Technologies*, pages 208–221. Springer, 2013.

[10] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[11] L. Chen, P. Koutris, and A. Kumar. Model-based pricing: Do not pay for more than what you learn! In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, page 1. ACM, 2017.

[12] L. Chen, P. Koutris, and A. Kumar. Model-based pricing for machine learning in a data marketplace. *arXiv preprint arXiv:1805.11450*, 2018.

[13] M. Chessa and P. Loiseau. A cooperative game-theoretic approach to quantify the value of personal data in networks. In *Proceedings of the 12th workshop on the Economics of Networks, Systems and Computation*, page 9. ACM, 2017.

[14] D. Dao, D. Alistarh, C. Musat, and C. Zhang. Databright: Towards a global exchange for decentralized data ownership and trusted computation. *arXiv preprint arXiv:1802.04780*, 2018.

[15] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[16] S. Deep, P. Koutris, and Y. Bidasaria. Qirana demonstration: real time scalable query pricing. *PVLDB*, 10(12):1949–1952, 2017.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[18] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[19] S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):325–327, 1976.

[20] S. S. Fatima, M. Wooldridge, and N. R. Jennings. A linear approximation method for the shapley value. *Artificial Intelligence*, 172(14):1673–1699, 2008.

[21] A. Ghorbani and J. Zou. Data shapley: Equitable valuation of data for machine learning. *arXiv preprint arXiv:1904.02868*, 2019.

[22] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[23] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[24] M. Hardt, E. Price, N. Srebro, et al. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323, 2016.

[25] J. Hays and A. A. Efros. Large-scale image geolocalization. In *Multimodal Location Estimation of Videos and Images*, pages 41–62. Springer, 2015.

[26] J. He, S. Kumar, and S.-F. Chang. On the difficulty of nearest neighbor search. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 41–48. Omnipress, 2012.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song. A demonstration of sterling: a privacy-preserving data marketplace. *PVLDB*, 11(12):2086–2089, 2018.

[29] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, B. Li, C. Zhang, D. Song, and C. Spanos. Towards efficient data valuation based on the shapley value. *AISTATS*, 2019.

[30] J. Kleinberg, C. H. Papadimitriou, and P. Raghavan. On the value of private information. In *Proceedings of the 8th conference on Theoretical aspects of rationality and knowledge*, pages 249–257. Morgan Kaufmann Publishers Inc., 2001.

[31] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

[32] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Querymarket demonstration: Pricing for online data markets. *PVLDB*, 5(12):1962–1965, 2012.

[33] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Toward practical query pricing with querymarket. In *proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 613–624. ACM, 2013.

[34] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. *Journal of the ACM (JACM)*, 62(5):43, 2015.

[35] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[36] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. *Communications of the ACM*, 60(12):79–86, 2017.

[37] C. Li and G. Miklau. Pricing aggregate queries in a data marketplace. In *WebDB*, pages 19–24, 2012.

[38] C. Li, S. Zhang, H. Zhang, L. Pang, K. Lam, C. Hui, and S. Zhang. Using the k-nearest neighbor algorithm for the classification of lymph node metastasis in gastric cancer. *Computational and mathematical methods in medicine*, 2012,

2012.

[39] B.-R. Lin and D. Kifer. On arbitrage-free pricing for general data queries. *PVLDB*, 7(9):757–768, 2014.

[40] S. Maleki. *Addressing the computational issues of the Shapley value with applications in the smart grid*. PhD thesis, University of Southampton, 2015.

[41] S. Maleki, L. Tran-Thanh, G. Hines, T. Rahwan, and A. Rogers. Bounding the estimation error of sampling-based shapley value approximation. *arXiv preprint arXiv:1306.4265*, 2013.

[42] T. P. Michalak, K. V. Aadithya, P. L. Szczepanski, B. Ravindran, and N. R. Jennings. Efficient computation of the shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research*, 46:607–650, 2013.

[43] D. M. Mount and S. Arya. Ann: library for approximate nearest neighbour searching. 1998.

[44] K. Ogawa, Y. Suzuki, and I. Takeuchi. Safe screening of non-support vectors in pathwise svm computation. In *International Conference on Machine Learning*, pages 1382–1390, 2013.

[45] R. Raskar, P. Vepakomma, T. Swedish, and A. Sharan. Data markets to support ai for all: Pricing, valuation and governance. *arXiv preprint arXiv:1905.06462*, 2019.

[46] L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[47] C. Siagian and L. Itti. Rapid biologically-inspired scene classification using features shared with visual attention. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):300–312, 2007.

[48] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[49] P. Upadhyaya, M. Balazinska, and D. Suciu. How to price shared optimizations in the cloud. *PVLDB*, 5(6):562–573, 2012.

[50] P. Upadhyaya, M. Balazinska, and D. Suciu. Price-optimal querying with data apis. *PVLDB*, 9(14):1695–1706, 2016.

[51] B. Woodworth, S. Gunasekar, M. I. Ohannessian, and N. Srebro. Learning non-discriminatory predictors. *arXiv preprint arXiv:1702.06081*, 2017.

[52] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333, 2013.

[53] Z. Zheng, Y. Peng, F. Wu, S. Tang, and G. Chen. Arete: On designing joint online pricing and reward sharing mechanisms for mobile data markets. *IEEE Transactions on Mobile Computing*, 2019.