Efficient Data Shapley for Weighted Nearest Neighbor Algorithms

Jiachen T. Wang[*] Princeton University

Prateek Mittal Princeton University

Ruoxi Jia[*] Virginia Tech

Abstract

This work aims to address an open problem in data valuation literature concerning the efficient computation of Data Shapley for weighted K nearest neighbor algorithm (WKNN-Shapley). By considering the accuracy of hard-label KNN with discretized weights as the utility function, we reframe the computation of WKNN-Shapley into a counting problem and introduce a quadratic-time algorithm, presenting a notable improvement from $O(N^K)$, the best result from existing literature. We develop a deterministic approximation algorithm that further improves computational efficiency while maintaining the key fairness properties of the Shapley value. Through extensive experiments, we demonstrate WKNN-Shapley's computational efficiency and its superior performance in discerning data quality compared to its unweighted counterpart.

1 INTRODUCTION

Data is the backbone of machine learning (ML) models, but not all data is created equally. In real-world scenarios, data often carries noise and bias, sourced from diverse origins and labeling processes Northcutt et al. (2021). Against this backdrop, data valuation emerges as a growing research field, aiming to quantify the quality of individual data sources for ML training. Data valuation techniques are critical in explainable ML to diagnose influential training instances and in data marketplaces for fair compensation. The importance of data valuation is highlighted by legislative efforts (Warner, 2019) and vision statements from leading tech companies (OpenAI, 2023). For instance, OpenAI listed "*how to fairly distribute the benefits the AI systems generate*" as an important question to be explored.

Data Valuation via the Shapley Value. Drawing on cooperative game theory, the technique of using the Shapley value for data valuation was pioneered by Ghorbani and Zou (2019); Jia et al. (2019b). The Shapley value is a renowned solution concept in game theory for fair profit attribution (Shapley, 1953). In the context of data valuation, individual data points or sources are regarded as "players" in a cooperative game, and *Data Shapley* refers to the suite of data valuation techniques that use the Shapley value as the contribution measure for each data owner.

KNN-Shapley. Despite offering a principled approach to data valuation with a solid theoretical foundation, the exact calculation of the Shapley value has the time complexity of $O(2^N)$ in general (Deng and Papadimitriou, 1994), where N refers to the number of data points/sources. Fortunately, Jia et al. (2019a) discovered an efficient $O(N \log N)$ algorithm to compute the exact Data Shapley for unweighted K-Nearest Neighbors (KNN), one of the oldest yet still popular ML algorithms. *KNN-Shapley* refers to the technique of assessing data value for any learning algorithms based on KNN's Data Shapley score. Here, KNN serves as a proxy model for the original, perhaps complicated learning algorithm. KNN-Shapley can be applied to large, high-dimensional datasets by calculating the value scores on the features extracted from neural network embeddings. Due to its superior efficiency and effectiveness in discerning data quality, KNN-Shapley has become one of the most popular data valuation techniques (Pandl et al., 2021).

Open question from Jia et al. (2019a): efficient computation of weighted KNN-Shapley. While Jia et al. (2019a) showed unweighted KNN-Shapley can be computed efficiently, they did not develop a practical algorithm for the more general weighted KNN-Shapley (WKNN-Shapley). Despite presenting a polynomial-time algorithm, its computational complexity of $O(N^K)$

[*]Correspondence to **Jiachen T. Wang** and **Ruoxi Jia** (email: tianhaowang@princeton.edu, ruoxijia@vt.edu).

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

becomes impractical even for modest K, such as 5. Closing this efficiency gap is important, especially given the inherent advantages and wider application of weighted KNN. Compared with the unweighted counterpart, weighted KNN considers the distance between data points, assigning varying levels of importance to neighbors based on their proximity to the query. Consequently, the Data Shapley of weighted KNN can potentially better discriminate between low and high-quality data points. Moreover, weighted KNN is applied more widely in practice. For example, recent research discovered weighted KNN's capability to improve language model's performance (Khandelwal et al., 2019).

Our contributions are summarized as follows:

Making KNN Configurations "Shapley-friendly" (Section 3). Our preliminary investigations suggest that improving the computational efficiency of WKNN-Shapley for soft-label KNN classifiers with continuous weight values (the setting considered in Jia et al. (2019a)), poses considerable challenges. Consequently, we make necessary modifications to the specific KNN classifiers' configuration and shift our focus to *hard-label* KNN classifiers with *discrete weight values*. The justification for these changes and their practical relevance is detailed in Section 3. In particular, discretizing weights does not change the data value score significantly even with few bits. We emphasize that making proper tweaks to the problem setup is important for developing an efficient Shapley computation algorithm, a strategy frequently adopted in literature (Dall'Aglio et al., 2019).

A quadratic-time algorithm for computing exact WKNN-Shapley (Section 4.1). Given the adjusted "Shapley-friendly" configurations of the weighted KNN, we can reframe the Shapley value computation as a counting problem. We develop an algorithm with a quadratic runtime for solving the counting problem and computing the exact WKNN-Shapley, which greatly improves the baseline $O(N^K)$ algorithm.

A subquadratic-time deterministic approximation algorithm that preserves fairness properties (Section 4.2). To further improve the computational efficiency, we propose a deterministic approximation algorithm by making minor changes to the exact WKNN-Shapley implementation. In particular, our approximation algorithm retains the crucial fairness properties of the original Shapley value.

Empirical Evaluations (Section 5). We experiment on benchmark datasets and assess the efficiency and efficacy of our exact and approximation algorithms for WKNN-Shapley. Here are the key takeaways: **(1)** Our exact and approximation algorithm for WKNN-Shapley significantly improves computational efficiency compared to the baseline exact algorithm and Monte Carlo approximation, respectively. **(2)** WKNN-Shapley outperforms the unweighted KNN-Shapley in discerning data quality for critical downstream tasks such as detecting mislabeled or noisy data. Remarkably, the approximated WKNN-Shapley matches the performance of the exact WKNN-Shapley on many benchmark datasets, attributable to its deterministic nature and the preservation of fairness properties.

Overall, with proper changes to KNN configurations, we show that WKNN-Shapley can be efficiently calculated and approximated. This facilitates its wider adoption, offering a more effective data valuation method compared to unweighted KNN-Shapley.

2 PRELIMINARIES

In this section, we formalize the setup of data valuation for ML, and revisit relevant techniques.

Setup & Goal. Given a labeled dataset $D := \{z_i\}_{i=1}^N$ where each data point $z_i := (x_i, y_i)$, data valuation aims to assign a score to each training data point z_i, reflecting its importance for the trained ML model's performance. Formally, we seek a score vector $(\phi_{z_i})_{i=1}^N$ where each $\phi_{z_i} \in \mathbb{R}$ represents the "value" of z_i.

2.1 Data Shapley

The Shapley value (SV) (Shapley, 1953), originating from game theory, stands out as a distinguished method for equitably distributing total profit among all participating players. Before diving into its definition, we first discuss a fundamental concept: the *utility function*.

Utility Function. A *utility function* maps an input dataset to a score indicating the utility of the dataset for model training. Often, this function is chosen as the validation accuracy of a model trained on the given dataset. That is, given a training set S, the utility function $v(S) := \texttt{ValAcc}(\mathcal{A}(S))$, where $\mathcal{A}$ represents a learning algorithm that trains a model on dataset S, and $\texttt{ValAcc}(\cdot)$ is a function assessing the model's performance, e.g., its accuracy on a validation set.

Definition 1 (Shapley value (Shapley, 1953))**.** *Let $v(\cdot)$ denote a utility function and D represent a training set of N data points. The Shapley value, $\phi_z(v)$, assigned to a data point $z \in D$ is defined as* $\phi_z(v) := \frac{1}{N}\sum_{k=1}^{N}\binom{N-1}{k-1}^{-1}\sum_{S\subseteq D_{-z},|S|=k-1}[v(S\cup\{z\})-v(S)]$ *where $D_{-z} = D \setminus \{z\}$.*

In simple terms, the Shapley value is a weighted average of the *marginal contribution* $v(S\cup\{z\})-v(S)$, i.e., the utility change when the point z is added to different Ss. For simplicity, we often write ϕ_z when the utility

function is clear from the context. The Shapley value uniquely satisfies several important axioms, including key fairness requirements like the *null player* and *symmetry* axioms, lending justification to its popularity. See Appendix A for detailed axiom definitions.

2.2 KNN-Shapley

A well-known challenge of using the Shapley value is that its exact calculation is computationally infeasible in general, as it usually requires evaluating $v(S)$ for all possible subsets $S \subseteq D$. A surprising result in Jia et al. (2019a) showed that when the learning algorithm $\mathcal{A}$ is *unweighted KNN*, there exists a highly efficient algorithm for computing its exact Data Shapley score.

K Nearest Neighbor Classifier. Given a validation data point $z^{(\text{val})} = (x^{(\text{val})}, y^{(\text{val})})$ and a distance metric $d(\cdot, \cdot)$, we sort the training set $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ according to their distance to the validation point $d(x_i, x^{(\text{val})})$ in non-descending order. Throughout the paper, we assume that $d(x_i, x^{(\text{val})}) \leq d(x_j, x^{(\text{val})})$ for any $i \leq j$ unless otherwise specified. A KNN classifier makes a prediction for the query $x^{(\text{val})}$ based on the (weighted) majority voting among $x^{(\text{val})}$'s K nearest neighbors in the training set. **Weight of data point:** in KNN, each data point z_i is associated with a weight w_i. The weight is usually determined based on the distance between x_i and the query $x^{(\text{val})}$. For example, a popular weight function is RBF kernel $w_i := \exp(-d(x_i, x^{(\text{val})}))$. If w_i is the same for all z_is, it becomes *unweighted* KNN.

Jia et al. (2019a) considers the utility function for the weighted, *soft-label* KNN:

$$v(S; z^{(\text{val})}) := \frac{\sum_{j=1}^{\min(K,|S|)} w_{\alpha_{x^{(\text{val})}}^{(S,j)}} \mathbf{1}\left[y_{\alpha_{x^{(\text{val})}}^{(S,j)}} = y^{(\text{val})}\right]}{\sum_{j=1}^{\min(K,|S|)} w_{\alpha_{x^{(\text{val})}}^{(S,j)}}} \tag{1}$$

where $\alpha_{x^{(\text{val})}}^{(S,j)}$ denotes the index (among D) of jth closest data point in S to $x^{(\text{val})}$. "Soft-label" refers to the classifiers that output the confidence scores. The main result in Jia et al. (2019a) shows that for *unweighted* KNN, we can compute the *exact* Shapley value $\phi_{z_i}\left(v(\cdot; z^{(\text{val})})\right)$ for *all* $z_i \in D$ within a total runtime of $O(N \log N)$ (see Appendix B for details).

Remark 1. *In practice, the model performance is assessed based on a validation set $D^{(\text{val})}$. After computing $\phi_{z_i}\left(v(\cdot; z^{(\text{val})})\right)$ for each $z^{(\text{val})} \in D^{(\text{val})}$, one can compute the Shapley value corresponding to the utility function on the full validation set $v(S; D^{(\text{val})}) := \sum_{z^{(\text{val})} \in D^{(\text{val})}} v(S; z^{(\text{val})})$ by simply taking the sum $\phi_{z_i}\left(v(\cdot; D^{(\text{val})})\right) = \sum_{z^{(\text{val})} \in D^{(\text{val})}} \phi_{z_i}\left(v(\cdot; z^{(\text{val})})\right)$ due to the* linearity *property of the Shapley value.*

Remark 2. *In alignment with the existing literature (Jia et al., 2019a; Wang et al., 2023), our discussion of the time complexity of KNN-Shapley refers to the total runtime needed to calculate* all *data value scores $\left(\phi_{z_1}(v(\cdot; z^{(\text{val})})), \ldots, \phi_{z_N}(v(\cdot; z^{(\text{val})}))\right)$, given that standard applications of data valuation, such as profit allocation and bad data detection, all require computing the data value scores for* all *data points in the training set. Furthermore, the stated runtime is with respect to $v(\cdot; z^{(\text{val})})$, and the overall runtime with respect to $v(\cdot; D^{(\text{val})})$ will be multiplied by the size of validation set $D^{(\text{val})}$. Runtime is typically presented this way because SV computations with respect to different $v(\cdot; z^{(\text{val})})$ are independent, readily benefit from parallel computing.*

Since its introduction, KNN-Shapley has rapidly gained popularity in data valuation for its efficiency and effectiveness, and is being advocated as the *'most practical technique for effectively evaluating large-scale data'* in recent studies (Pandl et al., 2021; Karlaš et al., 2022).

2.3 Baseline Algorithm for Computing and Approximating WKNN-Shapley

Jia et al. (2019a) developed an efficient $O(N \log N)$ algorithm to calculate the exact unweighted KNN-Shapley. However, when it comes to the more general weighted KNN-Shapley, only an $O(N^K)$ algorithm is given. While still in polynomial time (if K is considered a constant), the runtime is impractically large even for small K (e.g., 5). Here, we review the high-level idea of the baseline algorithms from Jia et al. (2019a).

An $O(N^K)$ algorithm for exact WKNN-Shapley computation. From Definition 1, the Shapley value for z_i is a weighted average of the *marginal contribution (MC)* $v(S \cup \{z_i\}) - v(S)$; hence, we only need to study those S whose utility might change due to the inclusion of z_i. For KNN, those are the subsets S where z_i is within the K nearest neighbors of $x^{(\text{val})}$ after being added into S. Note that for KNN, the utility of any S only depends on the K nearest neighbors of $x^{(\text{val})}$ in S. Given that there are only $\sum_{j=0}^{K} \binom{N}{j}$ unique subsets of size $\leq K$, we can simply query the MC value $v(S \cup \{z_i\}) - v(S)$ for all S of size $\leq K$. For any larger S, the MC must be the same as its subset of K nearest neighbors. We can then compute the Shapley value as a weighted average of these MC values by counting the number of subsets that share the same MC values through simple combinatorial analysis. Such an algorithm results in the runtime of $\sum_{j=0}^{K} \binom{N}{j} = O(N^K)$. The algorithm details can be found in Appendix B.

Monte Carlo Approximation. Given the large runtime of this exact algorithm, Jia et al. (2019a) further proposes an approximation algorithm based on Monte Carlo techniques. However, Monte Carlo-based approx-

imation is randomized and may not preserve the fairness property of the exact Shapley value. Additionally, the sample complexity of the Monte Carlo estimator is derived from concentration inequalities, which, while suitable for asymptotic analysis, may provide loose bounds in practical applications.

3 MAKING KNN CONFIGURATIONS SHAPLEY-FRIENDLY

We point out the major challenges associated with directly improving the computational efficiency for the soft-label KNN configuration considered in Jia et al. (2019a), and propose proper changes that enable more efficient algorithms for computing WKNN-Shapley.

Challenge #1: weights normalization term. The key principle behind the $O(N \log N)$ algorithm for unweighted KNN-Shapley from Jia et al. (2019a) is that, the MC can only take few distinct values. For example, for any $|S| \geq K$, we have $v(S \cup \{z_i\}) - v(S) = \frac{1}{K}\left(\mathbf{1}[y_i = y^{(\text{val})}] - \mathbf{1}[y_{\alpha_{x^{(\text{val})}}^{(S,K)}} = y^{(\text{val})}]\right)$. To avoid the task of evaluating $v(S)$ for all $S \subseteq D$, one can just count the subsets $S \subseteq D \setminus \{z_i\}$ such that z_i is among the K nearest neighbors of $x^{(\text{val})}$ in $S \cup \{z_i\}$, as well as the subsets share the same Kth nearest neighbor to $z^{(\text{val})}$. However, for weighted soft-label KNN with the utility function in (1), there is little chance that any of two $v(S_1 \cup \{z_i\}) - v(S_1)$ and $v(S_2 \cup \{z_i\}) - v(S_2)$ can have the same value due to the weights normalization term $\left(\sum_{j=1}^{K} w_{\alpha_{x^{(\text{val})}}^{(S,j)}}\right)^{-1}$ (note that this term is $1/K$, a constant, for unweighted setting). **Solution #1: hard-label KNN.** In this work, we instead consider the utility function for weighted *hard-label* KNN. "Hard-label" refers to the classifiers that output the predicted class instead of the confidence scores (see (2) in Section 4). In practice, user-facing applications usually only output a class prediction instead of the entire confidence vector. More importantly, hard-label KNN's prediction only depends on the weight comparison between different classes, and hence its utility function does not have a normalization term.

Challenge #2: continuous weights. If the weights are on the continuous space, there will be infinitely many possibilities of weighted voting scores of the K nearest neighbors. This makes it difficult to analyze which pairs of S_1, S_2 share the same MC value. **Solution #2: discretize weights.** Therefore, we consider a more tractable setting where the weights lie in a discrete space. Such a change is reasonable since the weights are stored in terms of finite bits (and hence in the discrete space) in practice. Moreover, rounding is a deterministic operation and does not reverse the original order of weights. In Appendix C, we show that the Shapley value computed based on the discrete weights has the same ranking order compared with the Shapley value computed on the continuous weights (it might create ties but will not reverse the original order). In Appendix G.2, we empirically verify that weight discretization does not cause a large deviation in the Shapley value.

We emphasize that, proper adjustments to the underlying utility function are important for efficient Shapley computation, and such a strategy is frequently applied in game theory literature (Dall'Aglio et al., 2019).

4 DATA SHAPLEY FOR WEIGHTED KNN

In this section, we develop efficient solutions for computing and approximating Data Shapley scores for weighted, hard-label KNN binary classifiers, where the weight values used in KNN are discretized. Without loss of generality, in this paper we assume every weight $w_i \in [0, 1]$.[1] We use $\mathbf{W}$ to denote the discretized space of $[0, 1]$, where we create 2^b equally spaced points within the interval when we use b bits for discretization. We denote $W := |\mathbf{W}| = 2^b$ the size of the weight space.

Utility Function for Weighted Hard-Label KNN Classifiers. The utility function of weighted hard-label KNN, i.e., the correctness of weighted KNN on the queried example $z^{(\text{val})}$, can be written as

$$v(S; z^{(\text{val})}) = \mathbf{1}\left[y^{(\text{val})} \in \operatorname*{argmax}_{c \in \mathbf{C}} \sum_{j=1}^{\min(K,|S|)} w_{\alpha_{x^{(\text{val})}}^{(S,j)}} \times \mathbf{1}[y_{\alpha_{x^{(\text{val})}}^{(S,j)}} = c]\right] \tag{2}$$

where $\mathbf{C} = \{1, \ldots, C\}$ is the space of classes, and C is the number of classes.[2] We omit the input of $z^{(\text{val})}$ and simply write $v(S)$ when the validation point is clear from the context. For KNN binary classifier, we can rewrite the utility function in a more compact form:

$$v(S) = \mathbf{1}\left[\sum_{j=1}^{\min(K,|S|)} \widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,j)}} \geq 0\right] \text{ where } \widetilde{w}_j := \begin{cases} w_j & y_j = y^{(\text{val})} \\ -w_j & y_j \neq y^{(\text{val})} \end{cases} \tag{3}$$

For ease of presentation, we present the algorithms for KNN binary classifier here, and defer the extension to multi-class classifier to Appendix E.

[1] If it does not hold one can simply normalize the weights to $[0, 1]$ and the KNN classifier remains the same.

[2] If multiple classes have the same top counts, we take the utility as 1 as long as $y^{(\text{val})}$ is among the majority classes.

4.1 Exact WKNN-Shapley Calculation

4.1.1 Computing SV is a Counting Problem

Given that the Shapley value is a weighted average of the marginal contribution $v(S \cup \{z_i\}) - v(S)$, we first study the expression of $v(S \cup \{z_i\}) - v(S)$ for a fixed subset $S \subseteq D \setminus \{z_i\}$ with the utility function in (3).

Theorem 2. *For any data point $z_i \in D$ and any subset $S \subseteq D \setminus \{z_i\}$, the marginal contribution has the expression as follows:*

$$v(S \cup \{z_i\}) - v(S) = \begin{cases} 1 & \text{if } y_i = y^{(\text{val})}, \texttt{Cond}_{KNN}, \texttt{Cond}_{0to1} \\ -1 & \text{if } y_i \neq y^{(\text{val})}, \texttt{Cond}_{KNN}, \texttt{Cond}_{1to0} \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

where

$$\texttt{Cond}_{KNN} := z_i \text{ is within } K \text{ nearest neighbors of } x^{(\text{val})} \text{ among } S \cup \{z_i\}$$

$$\texttt{Cond}_{0to1} := \begin{cases} \sum_{z_j \in S} \widetilde{w}_j \in [-\widetilde{w}_i, 0) & \text{if } |S| \leq K-1 \\ \sum_{j=1}^{K-1} \widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,j)}} \in \left[-w_i, -\widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,K)}}\right) & \text{if } |S| \geq K \end{cases}$$

$$\texttt{Cond}_{1to0} := \begin{cases} \sum_{z_j \in S} \widetilde{w}_j \in [0, -\widetilde{w}_i) & \text{if } |S| \leq K-1 \\ \sum_{j=1}^{K-1} \widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,j)}} \in \left[-\widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,K)}}, -w_i\right) & \text{if } |S| \geq K \end{cases}$$

In words, the condition $\texttt{Cond}_{KNN}$ means that z_i should be among the K nearest neighbors to the query sample when it is added to S. The conditions $\texttt{Cond}_{0to1}$ and $\texttt{Cond}_{1to0}$ cover situations where adding z_i to the set S changes the prediction of the weighted KNN classifiers. In greater detail, $\texttt{Cond}_{0to1}$ captures the condition for which incorporating z_i shifts the "*effective sum of signed weights $\widetilde{w}$*" from a negative to a non-negative value, thereby incrementing the utility from 0 to 1. $\texttt{Cond}_{1to0}$ can be interpreted similarly. From Theorem 2 and the formula of the Shapley value (Definition 1), we can reframe the problem of computing hard-label WKNN-Shapley as a counting problem. Specifically, this involves counting the quantity defined as follows:

Definition 3. *Let $\texttt{G}_{i,\ell}$ denote the count of subsets $S \subseteq D \setminus z_i$ of size ℓ that satisfy **(1)** $\texttt{Cond}_{KNN}$, and **(2)** $\texttt{Cond}_{0to1}$ if $y_i = y^{(\text{val})}$, or $\texttt{Cond}_{1to0}$ if $y_i \neq y^{(\text{val})}$.*

Theorem 4. *For a weighted, hard-label KNN binary classifier using the utility function given by (3), the Shapley value of a data point z_i can be expressed as:*

$$\phi_{z_i} = \frac{2\mathbf{1}[y_i = y^{(\text{val})}] - 1}{N} \sum_{\ell=0}^{N-1} \binom{N-1}{\ell}^{-1} \texttt{G}_{i,\ell} \tag{5}$$

Figure 1 illustrates the counting problem we try to solve here.

4.1.2 Dynamic Programming Solution for Computing $\texttt{G}_{i,\ell}$

The multiple, intricate conditions wrapped within $\texttt{G}_{i,\ell}$'s definition can pose a formidable challenge for direct and efficient counting. The main rationale behind our solution is to break down the complex counting problems into smaller, more manageable subproblems, thereby making them amenable to algorithmic solutions like dynamic programming. Before delving into the specifics of the algorithm, we introduce an intermediary quantity, $\texttt{F}_i$, that becomes the building block of our dynamic programming formulation.

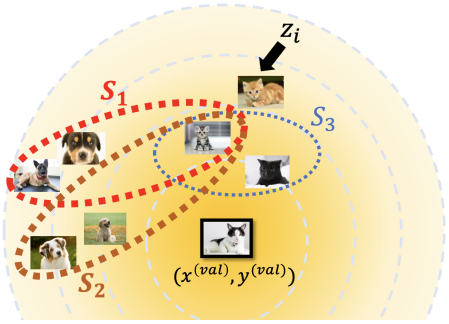

Figure 1: Illustration of the subsets targeted in the counting problem. When $K = 3$, both S_1 and S_2 have a utility of 0 as both of them contain 2 dogs and 1 cat. Adding z_i to S_1 and S_2 alters the 3 nearest neighbors to the query image $x^{(\text{val})}$, which now contains 1 dog and 2 cats, raising the utility to 1. In contrast, S_3's utility remains unchanged with the addition of z_i since it solely contains cat images. To compute WKNN-Shapley of z_i, we count the subsets S where adding z_i changes its utility, as seen with S_1 and S_2.

Definition 5. *Let $\texttt{F}_i[m, \ell, s]$ denote the count of subsets $S \subseteq D \setminus \{z_i\}$ of size ℓ that satisfy **(1)** $\texttt{Cond}_{KNN}$, as well as the following conditions: **(2)** Within S, the data point x_m is the $\min(\ell, K)$-th closest to the query example $x^{(\text{val})}$, **(3)** $\sum_{j=1}^{\min(\ell,K-1)} \widetilde{w}_{\alpha_{x^{(\text{val})}}^{(S,j)}} = s$.*

We can relate this auxiliary quantity to our desired $\texttt{G}_{i,\ell}$ as follows:

Theorem 6 (Relation between $\texttt{G}_{i,\ell}$ and $\texttt{F}_i$)**.** *For $y_i = y^{(\text{val})}$, we can compute $\texttt{G}_{i,\ell}$ from $\texttt{F}_i$ as follows:*

$$\texttt{G}_{i,\ell} = \begin{cases} \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, 0)} \texttt{F}_i[m, \ell, s] & \text{for } \ell \leq K-1, \\ \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \texttt{F}_i[m, \ell, s] & \text{for } \ell \geq K. \end{cases} \tag{6}$$

For $y_i \neq y^{(\text{val})}$, we have:

$$\texttt{G}_{i,\ell} = \begin{cases} \sum_{m \in [N] \setminus i} \sum_{s \in [0, -\widetilde{w}_i)} \texttt{F}_i[m, \ell, s] & \text{for } \ell \leq K-1, \\ \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_m, -\widetilde{w}_i)} \texttt{F}_i[m, \ell, s] & \text{for } \ell \geq K. \end{cases} \tag{7}$$

The auxiliary quantity $\texttt{F}_i$ thus serves as a pivot, allowing us to explore the search space of possible subsets S more systematically. We next exploit the computational advantage of $\texttt{F}_i$. Specifically, $\texttt{F}_i$ can be conveniently computed with (recursive) formulas, which

further enables us to compute $\mathtt{G}_{i,\ell}$ with reduced computational demand.

Theorem 7 (simplified version)**.** *For $\ell \leq K-1$, $\mathtt{F}_i[m,\ell,s]$ can be computed from $\mathtt{F}_i[t,\ell,\cdot]$ with $t \leq m-1$. For $\ell \geq K$, $\mathtt{F}_i[m,\ell,s]$ can be computed from $\mathtt{F}_i[t,K-1,\cdot]$ with $t \leq m-1$.*

Leveraging the results from Theorem 7, a direct method for calculating $\mathtt{G}_{i,\ell}$ for all $\ell \geq 1$ is as follows: we first use a recursive formula to compute $\mathtt{F}_i[\cdot,\ell,\cdot]$ for $\ell \leq K-1$, and then use an explicit formula to compute $\mathtt{F}_i[\cdot,\ell,\cdot]$ for $\ell \geq K$. With $\mathtt{F}_i$ being computed, we apply Theorem 6 to compute $\mathtt{G}_{i,\ell}$. This direct approach renders an $O(N^3)$ runtime, as it necessitates computing $\mathtt{F}_i[m,\ell,\cdot]$ for each of i, m, and ℓ in the range of $1,\ldots,N$.

Further Improvement of Efficiency Through Short-cut Formula. While the above direct approach offers a clear path, there exists a more optimized algorithm to expedite computational efficiency by circumventing the explicit calculations of $\mathtt{F}_i[\cdot,\ell,\cdot]$ for $\ell \geq K$. Specifically, we discover a short-cut formula that allows us to directly calculate the summation $\sum_{\ell=K}^{N-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}}$ from Theorem 4 once we obtain $\mathtt{F}_i[\cdot,K-1,\cdot]$.

Theorem 8. *For a weighted, hard-label KNN binary classifier using the utility function given by (3), the Shapley value of data point z_i can be expressed as:*

$$\phi_{z_i} = \mathtt{sign}(w_i)\left[\frac{1}{N}\sum_{\ell=0}^{K-1}\frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1)}^{N}\frac{\mathtt{R}_{i,m}}{m\binom{m-1}{K}}\right]$$

where

$$\mathtt{R}_{i,m} := \begin{cases} \sum_{t=1}^{m-1}\sum_{s\in[-\widetilde{w}_i,-\widetilde{w}_m)}\mathtt{F}_i[t,K-1,s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{t=1}^{m-1}\sum_{s\in[-\widetilde{w}_m,-\widetilde{w}_i)}\mathtt{F}_i[t,K-1,s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$$

Crucially, the $\mathtt{R}_{i,m}$ quantity in the above expression can be efficiently calculated using a clever caching technique. Based on the above findings, we can eliminate a factor of N in the final time complexity, thereby obtaining a *quadratic-time* algorithm to compute the exact WKNN-Shapley. The comprehensive pseudocode for the full algorithm can be found in Appendix D.2.

Theorem 9. *Algorithm 2 (in Appendix D.2) computes the exact WKNN-Shapley ϕ_{z_i} for all $i = 1,\ldots,N$ and achieves a total runtime of $O(WK^2N^2)$.*

Remark 3 (Runtime Dependency with K and W)**.** *While the time complexity in Theorem 9 also depends on K and W, these variables can be effectively treated as constants in our context. In our ablation study, we found that the error caused by weights discretization reduces quickly as the number of bits b for discretization grows. Hence, across all experiments in Section 5, we set the number of bits for discretization as $b = 3$ and therefore $W = 2^b = 8$. Additionally, the selection of K in KNN-Shapley literature commonly stabilizes around values of 5 or 10, irrespective of the dataset size (Jia et al., 2019a; Wang et al., 2023). This stability arises because, in the context of KNN classifiers, increasing K can easily result in underfitting even when N is large. Throughout our experiments, we fix K at 5. A detailed ablation study examining different choices of K and W is available in Appendix G.*

Remark 4 (Novelty in the derivation of WKNN-Shapley)**.** *Our derivation of WKNN-Shapley starts similarly to Jia et al. (2019a) by examining the marginal contribution, but **the methodologies significantly differ afterward.** Unweighted KNN-Shapley benefits from the simplicity of its utility function, allowing for a relatively straightforward derivation. Specifically, Jia et al. (2019a) plugs unweighted KNN's utility function into the formula of the difference of the Shapley values between two data points, and then simplifies the expression using known equalities in combinatorial analysis. In contrast, the same approach does not apply to WKNN-Shapley due to the complexity introduced by weights. Therefore, we develop a novel dynamic programming solution, which marks a substantial departure from the techniques used in Jia et al. (2019a). We would like to clarify that there is **no correlation between the "recursion" in Jia et al. (2019a) and our paper.** In Jia et al. (2019a), each ϕ_{z_j} is recursively computed from $\phi_{z_{j+1}}$. On the contrary, the computation of different data points' WKNN-Shapley scores ϕ_{z_j} is* independent *of each other. In our method, recursion is a fundamental component of dynamic programming to solve complex counting problems.*

4.2 Deterministic Approximation for Weighted KNN-Shapley

While the algorithm introduced in Section 4.1 for calculating the exact WKNN-Shapley achieves $O(N^2)$ runtime, a huge improvement from the original $O(N^K)$ algorithm from Jia et al. (2019a), there remains room for further improving the efficiency if we only require an approximation of the Shapley value. Contrary to the prevalent use of Monte Carlo techniques in existing literature, in this section, we develop a *deterministic* approximation algorithm for WKNN-Shapley.

Intuition. From Theorem 7, we know that in order to compute $\mathtt{F}_i[m,\ell,\cdot]$ with $\ell \leq K-1$, we only need to know $\mathtt{F}_i[t,\ell-1,\cdot]$ with $t \leq m-1$. Moreover, observe that the building blocks for $\mathtt{G}_{i,\ell}$ (or $\mathtt{R}_{i,m}$), $\sum_{s\in[-\widetilde{w}_i,0)}\mathtt{F}_i[t,\ell,s]$ (or $\sum_{s\in[-\widetilde{w}_i,-\widetilde{w}_m)}\mathtt{F}_i[t,K-1,s]$), can be quite small as it only takes the summation over a small range of the weight space. Hence, we can use $\widehat{\mathtt{F}}_i[m,\cdot,\cdot] = 0$ as an approximation for $\mathtt{F}_i[m,\cdot,\cdot]$ for all $m \geq M^\star + 1$ with some prespecified threshold $M^\star$. Similarly, we can use $\widehat{\mathtt{R}}_{i,m} = 0$ as an approximation for $\mathtt{R}_{i,m}$ for all

$m \geq M^\star + 1$. The resultant approximation for the Shapley value ϕ_{z_i} is stated as follows:

Definition 10. *We define the approximation* $\widehat{\phi}_{z_i}^{(M^\star)}$ *as*

$$\widehat{\phi}_{z_i}^{(M^\star)} := \texttt{sign}(w_i) \left[\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\widetilde{G}_{i,\ell}^{(M^\star)}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1)}^{M^\star} \frac{R_{i,m}}{m\binom{m-1}{K}} \right]$$

where

$$\widetilde{G}_{i,\ell}^{(M^\star)} := \begin{cases} \sum_{m=1}^{M^\star} \sum_{s \in [-\widetilde{w}_i, 0)} F_i[m, \ell, s] & \text{for } y_i = y^{(\text{val})} \\ \sum_{m=1}^{M^\star} \sum_{s \in [0, -\widetilde{w}_i)} F_i[m, \ell, s] & \text{for } y_i \neq y^{(\text{val})} \end{cases}$$

To calculate $\widehat{\phi}_{z_i}^{(M^\star)}$, we only need to compute $\texttt{F}_i[m, \cdot, \cdot]$ and $\texttt{R}_{i,m}$ for m from 1 to $M^\star$ instead of N, thereby reducing the runtime of Algorithm 2 to $O(NM^\star)$ with minimal modification to the exact algorithm's implementation.

Theorem 11. *Algorithm 3 (in Appendix D.3) computes the approximated WKNN-Shapley* $\widehat{\phi}_{z_i}^{(M^\star)}$ *for all* $z_i \in D$ *and achieves a total runtime of* $O(WK^2NM^\star)$.

In particular, when $M^\star = \sqrt{N}$, we can achieve the runtime of $O(N^{1.5})$. The selection of $M^\star$ is discussed in Remark 5. In the following, we derive the error bound and point out two nice properties of this approximation.

Theorem 12. *For any* $z_i \in D$, *the approximated Shapley value* $\widehat{\phi}_{z_i}^{(M^\star)}$ ***(1)*** *shares the same sign as* ϕ_{z_i}, ***(2)*** *ensures* $\left|\widehat{\phi}_{z_i}^{(M^\star)}\right| \leq |\phi_{z_i}|$, *and* ***(3)*** *has the approximation error bounded by* $\left|\widehat{\phi}_{z_i}^{(M^\star)} - \phi_{z_i}\right| \leq \varepsilon(M^\star)$ *where*

$$\varepsilon(M^\star) := \sum_{m=M^\star+1}^{N} \left(\frac{1}{m-K} - \frac{1}{m} \right) + \sum_{\ell=1}^{K-1} \frac{\binom{N}{\ell} - \binom{M^\star}{\ell}}{N\binom{N-1}{\ell}} = O\left(K/M^\star\right)$$

Leveraging the error bound $\varepsilon(M^\star)$ alongside the additional nice properties of $\widehat{\phi}_{z_i}^{(M^\star)}$ stated in Theorem 12, we can obtain a deterministic interval within which ϕ_{z_i} always resides. Specifically, when $y_i = y^{(\text{val})}$, we have $\phi_{z_i} \in \left[\widehat{\phi}_{z_i}^{(M^\star)}, \widehat{\phi}_{z_i}^{(M^\star)} + \varepsilon(M^\star)\right]$, and when $y_i \neq y^{(\text{val})}$, we have $\phi_{z_i} \in \left[\widehat{\phi}_{z_i}^{(M^\star)} - \varepsilon(M^\star), \widehat{\phi}_{z_i}^{(M^\star)}\right]$. Unlike the commonly used Monte Carlo method for approximating the Shapley value, which only offers a high-probability interval and allows for a failure possibility that the exact value might fall outside of it, our deterministic approximation ensures that the exact value is always within the corresponding interval.

The approximated WKNN-Shapley preserves the fairness axioms. The Shapley value's axiomatic properties, particularly the *symmetry* and *null player* axioms, are of great importance for ensuring fairness when attributing value to individual players (see Appendix A for the formal definitions of the two axioms). These fundamental axioms have fostered widespread adoption of the Shapley value. An ideal approximation of the Shapley value, therefore, should preserve at least the symmetry and null player axioms to ensure that the principal motivations for employing the Shapley value—fairness and equity—are not diminished. The prevalent Monte Carlo-based approximation techniques give randomized results and necessarily muddy the clarity of fairness axioms. In contrast, our deterministic approximation preserves both important axioms.

Theorem 13. *The approximated Shapley value* $\{\widehat{\phi}_{z_i}^{(M^\star)}\}_{z_i \in D}$ *satisfies symmetry and null player axiom.*

Remark 5 (**Selection of** $M^\star$)**.** *Ideally, we would like to pick the smallest* $M^\star$ *such that* $\varepsilon(M^\star)$ *is significantly smaller than* $|\phi_{z_i}|$ *for a significant portion of* z_i*s. However, determining a universally applicable heuristic for setting* $M^\star$ *is challenging due to the varying magnitude of* ϕ_{z_i} *across different datasets, which are difficult to anticipate. For example, in a case where all but one data point are "null players", that single data point will possess a value of* $v(D)$*, while all others will be valued at 0. On the other hand, if all data points are identical, each will receive a value of* $v(D)/N$*. Therefore, we suggest to select* $M^\star$ *in an adaptive way. Specifically, for each* $M^\star \in \{K+1, \ldots, N\}$*, we calculate* $(\widehat{\phi}_{z_i}^{(M^\star)})_{i \in [N]}$*, halting the computation when the magnitude of* $\varepsilon(M^\star)$ *is substantially smaller than (e.g.,* $< 10\%$*) the magnitude of* $\widehat{\phi}_{z_i}^{(M^\star)}$ *for a majority of* z_i*s. This approach does not increase the overall runtime since* $\widehat{\phi}_{z_i}^{(M^\star+1)}$ *can be easily computed from* $\widehat{\phi}_{z_i}^{(M^\star)}$ *and the additionally computed* $F_i[M^\star + 1, \cdot, \cdot]$*. Details of this approach are in Appendix D.4. Furthermore, we highlight that our deterministic approximation algorithm maintains the fairness properties of exact WKNN-Shapley. Hence, in practice, we can potentially use a smaller* $M^\star$ *and still get satisfactory performance in discerning data quality. Throughout our experiments in Section 5, we find that setting* $M^\star = \sqrt{N}$ *consistently works well across all benchmark datasets.*

5 NUMERICAL EXPERIMENTS

We systematically evaluate the performance of our WKNN-Shapley computation and approximation algorithms. Our experiments aim to demonstrate the following assertions: **(1)** Our exact and deterministic approximation algorithm for WKNN-Shapley significantly improves computational efficiency compared to the baseline $O(N^K)$ algorithm and Monte Carlo approximation, respectively. **(2)** Compared to unweighted KNN-Shapley, WKNN-Shapley **(2-1)** achieves a better performance in discerning data quality in several important downstream tasks including mislabeled and

noisy data detection, and **(2-2)** demonstrates more stable performance against different choices of Ks. **(3)** The approximated WKNN-Shapley, while being more efficient, consistently achieves performance comparable to the exact WKNN-Shapley on most of the benchmark datasets.

5.1 Runtime Comparison

We empirically assess the computational efficiency of our exact and deterministic approximation algorithms for WKNN-Shapley, comparing them to the $O(N^K)$ exact algorithm and the Monte Carlo approximation presented in Jia et al. (2019a). We examine various training data sizes N and compare the execution clock time of different algorithms at each N. In data size regimes where the baseline algorithms from Jia et al. (2019a) are infeasible to execute (> 10 hours), we fit a polynomial curve to smaller data size regimes and plot the predicted extrapolation for larger sizes.

Figure 2 shows that the exact algorithm from Jia et al. (2019a) requires $\geq 10^3$ hours to run even for $N = 100$, rendering it impractical for actual use. In contrast, our exact algorithm for computing WKNN-Shapley achieves a significantly better computational efficiency (e.g., **almost 10^6 times faster at $N = 10^5$**). Our deterministic approximation algorithm is compared to the Monte Carlo approximation from Jia et al. (2019a). For a fair comparison, both algorithms are aligned for the same theoretical error bounds. Note that the error bound for the Monte Carlo algorithm is a high-probability bound, subject to a small failure probability. In contrast, our error bound always holds, offering a more robust guarantee compared to the Monte Carlo technique. Nonetheless, from Figure 2 we can see that our deterministic approximation algorithm not only provides a stronger approximation guarantee but also achieves significantly greater efficiency (e.g., also **almost 10^6 times faster at $N = 10^5$**). This showcases the remarkable improvements of our techniques.

5.2 Discerning Data Quality

Due to the superior computational efficiency of our newly developed algorithms, WKNN-Shapley has become a practical data valuation technique for actual use. In this section, we evaluate its effectiveness in discerning data quality for common real-world applications. **Tasks:** we consider three applications that are commonly used for evaluating the performance of data valuation techniques in the prior works (Ghorbani and Zou, 2019; Kwon and Zou, 2022; Wang and Jia, 2023a): mislabeled and noisy data detection. Due to space constraints, we only present the results for mislabeled data detection here, and defer the results for

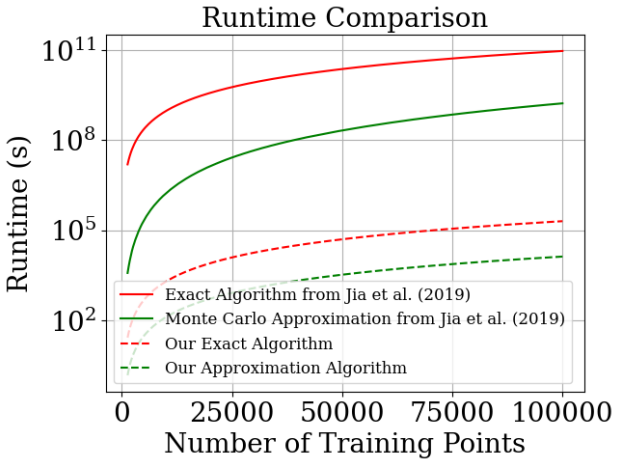

Figure 2: Runtime comparison between our exact and approximation algorithms for WKNN-Shapley in Section 4, and those from Jia et al. (2019a), across varying training data sizes N. We set $K = 5$ and the weights are discretized to 3-bit here. In Appendix G, we provide additional experiments on different Ks and bs. For our deterministic approximation algorithm, we set $M^\star = \sqrt{N}$ (so that the time complexity is $O(N^{1.5})$). For the Monte Carlo approximation from Jia et al. (2019a), we align the error bounds to be the same as ours for fair comparison; we set the failure probability for Monte Carlo method as $\delta = 0.1$. The plot shows the average runtime based on 5 independent runs.

the other tasks to Appendix G. Mislabeled data usually detrimentally impacts model performance. Hence, a reasonable data valuation technique should assign low values to these data points. In our experiments for mislabeled data detection, we randomly select 10% of the data points to flip their labels. **Baselines & Settings & Hyperparameters:** We evaluate the performance of both the exact and approximated WKNN-Shapley. We use ℓ_2 distance and the popular RBF kernel $w_i = \exp(-\left\|x_i - x^{(\text{val})}\right\|)$ to determine the weights of training points. We discretize the weights to 3 bits, as we find this level of precision offers a balance between good performance and computational efficiency, with a weight space size, W, of merely $2^3 = 8$. In Appendix G, we conduct ablation studies on the choice of the number of bits for discretization. For approximated WKNN-Shapley, we set $M^\star = \sqrt{N}$. Our primary baseline is the unweighted, soft-label KNN-Shapley from Jia et al. (2019a). Since our WKNN-Shapley corresponds to hard-label KNN, we also include unweighted, hard-label WKNN-Shapley in comparison for completeness. Note that it can be computed by simply setting the weights of all data points as a constant.

Results. We use AUROC as the performance metric on mislabeled data detection tasks. **Unweighted vs**

	Unweighted KNN-Shapley (Soft-label)	Unweighted KNN-Shapley (Hard-label, this work)	Exact WKNN-Shapley (this work)	Approximated WKNN-Shapley (this work)
2DPlanes	0.849	0.8	**0.884**	0.831
CPU	0.867	0.929	**0.956**	**0.956**
Phoneme	0.707	0.724	0.773	**0.778**
Fraud	0.556	0.547	**0.751**	0.596
Creditcard	0.698	0.676	**0.842**	0.747
Vehicle	0.689	0.724	0.8	**0.813**
Click	0.627	0.6	**0.751**	0.693
Wind	0.836	0.849	0.858	**0.88**
Pol	0.907	0.862	**1**	0.991
MNIST	0.724	0.471	0.831	**0.836**
CIFAR10	0.684	**0.76**	**0.76**	0.756
AGNews	0.953	0.978	**0.991**	0.988
DBPedia	0.968	0.902	**1**	**1**

Table 1: AUROC scores of different variants of KNN-Shapley for mislabeled data detection on benchmark datasets. The higher, the better.

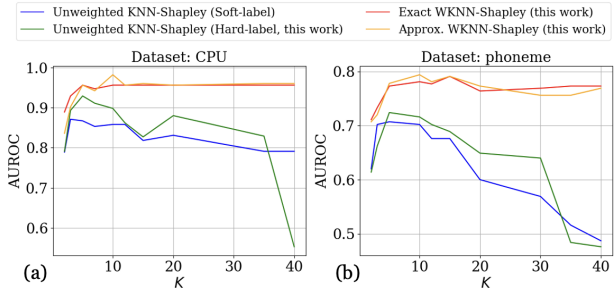

Figure 3: AUROC scores of different variants of KNN-Shapley for mislabeled data detection with different Ks. The higher the curve is, the better the method is.

weighted KNN-Shapley: Table 1 shows the AUROC scores across the 13 benchmark datasets we experimented on when $K = 5$. Notably, both exact and approximated WKNN-Shapley markedly outperform the unweighted KNN-Shapley (either soft-label or hard-label) across most datasets. This can likely be attributed to WKNN-Shapley's ability to more accurately differentiate between bad and good data based on the proximity to the queried example. In Appendix G.6, we present a qualitative study highlighting why WKNN-Shapley outperforms unweighted KNN-Shapley in discerning data quality. **Exact vs Approximated WKNN-Shapley:** From Table 1, we can see an encouraging result that the approximated WKNN-Shapley achieves performance comparable to (and sometimes even slightly better than) the exact WKNN-Shapley across the majority of datasets. This is likely attributable to its favored property in preserving the fairness properties of its exact counterpart. **Robustness to the choice of K:** In Figure 3, we show that, compared to unweighted KNN-Shapley, WKNN-Shapley maintains notably stable performance across various choices of K, particularly for larger values. This is because those benign data points—though within the K nearest neighbors of the query example and possessing different labels—may not receive a very low value due to their likely distant positioning from the query example. Conversely, unweighted KNN-Shapley tends to assign these benign points lower values.

6 CONCLUSION

In this study, we addressed WKNN-Shapley computation and approximation when using the accuracy of hard-label KNN with discretized weights as the utility function. Future work should explore polynomial time computation of exact Data Shapley for other learning algorithms. It is especially important to think about whether we can make some modifications to more complicated learning algorithms, e.g., neural networks, so that their exact Data Shapley can be computed efficiently.

Acknowledgements

This work was supported in part by the National Science Foundation under grants CNS-2131938, CNS-1553437, CNS-1704105, IIS-2312794, IIS-2313130, OAC-2239622, the ARL's Army Artificial Intelligence Innovation Institute (A2I2), the Office of Naval Research Young Investigator Award, the Army Research Office Young Investigator Prize, Schmidt DataX award, Princeton E-ffiliates Award, Amazon-Virginia Tech Initiative in Efficient and Robust Machine Learning, the Commonwealth Cyber Initiative, and a Princeton's Gordon Y. S. Wu Fellowship. We are grateful to Tinghao Xie, Tong Wu and Luxi He for their helpful feedback on the initial draft of this work. We are grateful to anonymous reviewers at AISTATS for their valuable feedback.

References

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, pages 722–735. Springer.

Belaid, M. K., Mekki, D. E., Rabus, M., and Hüllermeier, E. (2023). Optimizing data shapley interaction calculation from o (2^ n) to o (tn^ 2) for knn models. *arXiv preprint arXiv:2304.01224*.

Bian, Y., Rong, Y., Xu, T., Wu, J., Krause, A., and Huang, J. (2021). Energy-based learning for cooperative games, with applications to valuation problems in machine learning. *arXiv preprint arXiv:2106.02938*.

Courtnage, C. and Smirnov, E. (2021). Shapley-value data valuation for semi-supervised learning. In *Discovery Science: 24th International Conference, DS 2021, Halifax, NS, Canada, October 11–13, 2021, Proceedings 24*, pages 94–108. Springer.

Dal Pozzolo, A., Caelen, O., Johnson, R. A., and Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE.

Dall'Aglio, M., Fragnelli, V., and Moretti, S. (2019). Sometimes the computation of the shapley value is simple. *Handbook of the Shapley value*, 441.

Deng, X. and Papadimitriou, C. H. (1994). On the complexity of cooperative solution concepts. *Mathematics of operations research*, 19(2):257–266.

Ghorbani, A., Kim, M., and Zou, J. (2020). A distributional framework for data valuation. In *International Conference on Machine Learning*, pages 3535–3544. PMLR.

Ghorbani, A. and Zou, J. (2019). Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR.

Ghorbani, A., Zou, J., and Esteva, A. (2022). Data shapley valuation for efficient batch active learning. In *2022 56th Asilomar Conference on Signals, Systems, and Computers*, pages 1456–1462. IEEE.

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

Jia, R., Dao, D., Wang, B., Hubis, F. A., Gurel, N. M., Li, B., Zhang, C., Spanos, C. J., and Song, D. (2019a). Efficient task-specific data valuation for nearest neighbor algorithms. *Proceedings of the VLDB Endowment*.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gürel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. J. (2019b). Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR.

Just, H. A., Kang, F., Wang, T., Zeng, Y., Ko, M., Jin, M., and Jia, R. (2022). Lava: Data valuation without pre-specified learning algorithms. In *The Eleventh International Conference on Learning Representations*.

Karlaš, B., Dao, D., Interlandi, M., Li, B., Schelter, S., Wu, W., and Zhang, C. (2022). Data debugging with shapley importance over end-to-end machine learning pipelines. *arXiv preprint arXiv:2204.11131*.

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., and Lewis, M. (2019). Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kwon, Y. and Zou, J. (2022). Beta shapley: a unified and noise-reduced data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 8780–8802. PMLR.

Kwon, Y. and Zou, J. (2023). Data-oob: Out-of-bag estimate as a simple and efficient data value. *ICML*.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Liang, W., Liang, K.-H., and Yu, Z. (2021). Herald: An annotation efficient method to detect user disengagement in social conversations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3652–3665.

Liang, W., Zou, J., and Yu, Z. (2020). Beyond user self-reported likert scale ratings: A comparison model for automatic dialog evaluation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1363–1374.

Lin, J., Zhang, A., Lécuyer, M., Li, J., Panda, A., and Sen, S. (2022). Measuring the effect of training data on deep learning predictions via randomized experiments. In *International Conference on Machine Learning*, pages 13468–13504. PMLR.

Liu, Z., Just, H. A., Chang, X., Chen, X., and Jia, R. (2023). 2d-shapley: A framework for fragmented data valuation. *arXiv preprint arXiv:2306.10473*.

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

Northcutt, C. G., Athalye, A., and Mueller, J. (2021). Pervasive label errors in test sets destabilize machine learning benchmarks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

OpenAI, F. (2023). Planning for agi and beyond. `https://openai.com/blog/planning-for-agi-and-beyond`.

Pandl, K. D., Feiland, F., Thiebes, S., and Sunyaev, A. (2021). Trustworthy machine learning for health care: scalable data valuation with the shapley value. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 47–57.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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

Shim, D., Mai, Z., Jeong, J., Sanner, S., Kim, H., and Jang, J. (2021). Online class-incremental continual learning with adversarial shapley value. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9630–9638.

Wang, J. T. and Jia, R. (2023a). Data banzhaf: A robust data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 6388–6421. PMLR.

Wang, J. T. and Jia, R. (2023b). A note on" efficient task-specific data valuation for nearest neighbor algorithms". *arXiv preprint arXiv:2304.04258*.

Wang, J. T. and Jia, R. (2023c). A note on" towards efficient data valuation based on the shapley value". *arXiv preprint arXiv:2302.11431*.

Wang, J. T., Zhu, Y., Wang, Y.-X., Jia, R., and Mittal, P. (2023). Threshold knn-shapley: A linear-time and privacy-friendly approach to data valuation. *arXiv preprint arXiv:2308.15709*.

Wang, T., Rausch, J., Zhang, C., Jia, R., and Song, D. (2020). A principled approach to data valuation for federated learning. In *Federated Learning*, pages 153–167. Springer.

Wang, Z., Shan, X., Zhang, X., and Yang, J. (2021). N24news: A new dataset for multimodal news classification. *arXiv preprint arXiv:2108.13327*.

Warner, M. (2019). Warner & hawley introduce bill to force social media companies to disclose how they are monetizing user data. Government Document.

Wu, Z., Shu, Y., and Low, B. K. H. (2022). Davinz: Data valuation using deep neural networks at initialization. In *International Conference on Machine Learning*, pages 24150–24176. PMLR.

Yan, T. and Procaccia, A. D. (2021). If you like shapley then you'll love the core. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5751–5759.

Yeh, I.-C. and Lien, C.-h. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert systems with applications*, 36(2):2473–2480.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes, see Section 4.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes, see Section 4.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes.
   (b) Complete proofs of all theoretical results. Yes.
   (c) Clear explanations of any assumptions. Yes.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes.
   (b) The license information of the assets, if applicable. Yes.
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable.
   (d) Information about consent from data providers/curators. Not Applicable.
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable.

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable.
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable.
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable.

A EXTENDED RELATED WORKS

A.1 Data Shapley

Data Shapley is one of the first principled approaches to data valuation that became increasingly popular (Ghorbani and Zou, 2019; Jia et al., 2019b). Data Shapley is based on the *Shapley value*, a famous solution concept from game theory literature which is usually justified as the *unique* value notion satisfying the following four axioms:

(1) Null player: if $v(S \cup \{z_i\}) = v(S)$ for all $S \subseteq D \setminus \{z_i\}$, then $\phi_{z_i}(v) = 0$.

(2) Symmetry: if $v(S \cup \{z_i\}) = v(S \cup \{z_j\})$ for all $S \subseteq D \setminus \{z_i, z_j\}$, then $\phi_{z_i}(v) = \phi_{z_j}(v)$.

(3) Linearity: For utility functions v_1, v_2 and any $\alpha_1, \alpha_2 \in \mathbb{R}$, $\phi_{z_i}(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 \phi_{z_i}(v_1) + \alpha_2 \phi_{z_i}(v_2)$.

(4) Efficiency: for every v, $\sum_{z_i \in D} \phi_{z_i}(v) = v(D)$.

Since its introduction, numerous variants of Data Shapley have been developed (Jia et al., 2019a; Ghorbani et al., 2020; Wang et al., 2020; Bian et al., 2021; Kwon and Zou, 2022; Lin et al., 2022; Wu et al., 2022; Karlaš et al., 2022; Wang and Jia, 2023a,c; Liu et al., 2023; Wang et al., 2023), reflecting its effectiveness as a principled approach for quantifying data point contributions to ML model training. However, not all axioms mentioned above are considered necessary for a reasonable data valuation technique by the community. For example, Kwon and Zou (2022) argue that **(4) Efficiency** is not necessarily required for data valuation, and Yan and Procaccia (2021) argue that **(3) Linearity** is mainly a technical requirement and does not have a natural interpretation in the context of machine learning. That said, the **(1) Null player** and **(2) Symmetry** axioms are often seen as the "fairness axioms" and are deemed fundamental for any sound data valuation technique. Therefore, a reasonable Shapley approximation should uphold these two axioms to ensure the Shapley value's core advantages—fairness and equity—are not diminished. The deterministic approximation algorithm we develop in Section 4.2 meets these criteria.

A.2 KNN-Shapley

The computation of the Shapley value is notoriously resource-intensive. To the best of our knowledge, unweighted KNN stands as the *only* frequently deployed ML model where the exact Data Shapley can be efficiently computed. Due to its exceptional computational efficiency coupled with its capability to discern data quality, KNN-Shapley has become one of the most popular and practical data valuation techniques. For instance, Ghorbani et al. (2022) extends KNN-Shapley to active learning, Shim et al. (2021) applies it in a continual learning setting. Additionally, studies such as Liang et al. (2020, 2021) have leveraged KNN-Shapley to eliminate ambiguous samples in NLP tasks, and Courtnage and Smirnov (2021) has endorsed its use in semi-supervised learning data valuation. Belaid et al. (2023) extends the analysis of KNN-Shapley to the calculation to the Shapley interaction index. KNN-Shapley has also shown its practicality in real-world scenarios. For example, Pandl et al. (2021) shows that KNN-Shapley is the *only* practical data valuation technique for valuing large amounts of healthcare data. (Wang et al., 2023) considers a simple variant of unweighted KNN termed *Threshold KNN*, and develops an alternative of KNN-Shapley that can be easily incorporated with differential privacy. Both Wang et al. (2023) and our work demonstrate the importance of proper adjustments to the underlying KNN's configuration (i.e., the utility function) in developing new data valuation techniques with desired properties (computational efficiency, privacy compatibility, etc.).

All the studies mentioned above focus on *unweighted* KNN. On the other hand, *weighted* KNN incorporates more information about the underlying dataset. Consequently, the Data Shapley score derived from weighted KNN might offer a better assessment of individual data point quality (as demonstrated in our experiments).

Finally, we note that some alternative data valuation techniques are as efficient as KNN-Shapley (Just et al., 2022; Kwon and Zou, 2023). However, these methods lack a formal theoretical justification as the Shapley value-based approaches.

B ADDITIONAL BACKGROUND OF KNN-SHAPLEY

Notation Review. Recall that we use $\alpha_{x^{(\mathrm{val})}}^{(S,j)}$ denotes the index (among D) of jth closest data point in S to $x^{(\mathrm{val})}$.

B.1 Unweighted KNN-Shapley

The unweighted KNN-Shapley was originally proposed in Jia et al. (2019a) and was later refined in Wang and Jia (2023b). Specifically, Wang and Jia (2023b) considers the utility function for unweighted, soft-label KNN on a validation point $z^{(\mathrm{val})}$:

$$v(S; z^{(\mathrm{val})}) := \frac{\sum_{j=1}^{\min(K,|S|)} \mathbf{1}[y_{\alpha_{x^{(\mathrm{val})}}^{(S,j)}} = y^{(\mathrm{val})}]}{\min(|S|, K)} \tag{8}$$

which is a special case of the utility function for weighted KNN in (1). It can be interpreted as the probability of a soft-label KNN classifier in predicting the correct label for a validation point $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})}) \in D^{(\mathrm{val})}$. When $|S| = 0$, $v_{z^{(\mathrm{val})}}(S)$ is set to the accuracy by random guessing (i.e., $v(\emptyset) = 1/\texttt{\#class}$). Here is the main result of Wang and Jia (2023b):

Theorem 14 (KNN-Shapley (Wang and Jia, 2023b))**.** *Consider the utility function in (8). Given a validation data point $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})})$ and a distance metric $d(\cdot,\cdot)$, if we sort the training set $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ according to $d(x_i, x^{(\mathrm{val})})$ in ascending order, then the Shapley value of each data point ϕ_{z_i} corresponding to utility function $v_{z^{(\mathrm{val})}}$ can be computed recursively as follows:*

$$\phi_{z_N} = \frac{\mathbf{1}[N \geq 2]}{N}\left(\mathbf{1}[y_N = y^{(\mathrm{val})}] - \frac{\sum_{i=1}^{N-1}\mathbf{1}[y_i = y^{(\mathrm{val})}]}{N-1}\right)\left(\sum_{j=1}^{\min(K,N)-1}\frac{1}{j+1}\right) + \frac{1}{N}\left(\mathbf{1}[y_N = y^{(\mathrm{val})}] - \frac{1}{C}\right)$$

$$\phi_{z_i} = \phi_{z_{i+1}} + \frac{\mathbf{1}[y_i = y^{(\mathrm{val})}] - \mathbf{1}[y_{i+1} = y^{(\mathrm{val})}]}{N-1}\left[\sum_{j=1}^{\min(K,N)}\frac{1}{j} + \frac{\mathbf{1}[N \geq K]}{K}\left(\frac{\min(i,K)\cdot(N-1)}{i} - K\right)\right]$$

where C denotes the number of classes for the classification task.

The computation of all unweighted KNN-Shapley $(\phi_{z_1}, \ldots, \phi_{z_N})$ can be achieved in $O(N \log N)$ runtime in total, as the runtime is dominated by the sorting data points in D.

B.2 Baseline Algorithm for Computing and Approximating WKNN-Shapley

Exact Computation Algorithm. Jia et al. (2019a) shows that the Data Shapley for weighted KNN can be computed exactly with a runtime of $O(N^K)$. The high-level idea, as described in Section 2.3 in the maintext, is that we only need to consider evaluating $v(S)$ for those S where the addition of the target data point z_i may change the prediction of KNN. Moreover, there are at most $O(N^K)$ such Ss. For completeness, we state the specific expression for computing the exact WKNN-Shapley from Jia et al. (2019a). We note that the original theorem statement in Jia et al. (2019a) has minor errors and we also fix it here.

Theorem 15 (Jia et al. (2019a))**.** *Consider the utility function in (1). Let $B_k(i) = \{S : |S| = k, z_i \notin S, S \subseteq D\}$. Let $r(\cdot)$ be a function that maps the set of training data to their ranks of similarity to $x^{(\mathrm{val})}$. Then, the Shapley value ϕ_{z_i} of each training point z_i can be calculated recursively as follows:*

$$\phi_{z_N} = \frac{1}{N}\sum_{k=0}^{K-1}\frac{1}{\binom{N-1}{k}}\sum_{S\in B_k(z_N)}[v(S\cup\{z_N\}) - v(S)] \tag{9}$$

$$\phi_{z_i} = \phi_{z_{i+1}} + \frac{1}{N-1}\sum_{k=0}^{N-2}\frac{1}{\binom{N-2}{k}}\sum_{S\in D_{i,k}} A_{i,k}\left[v(S\cup\{z_i\}) - v(S\cup\{z_{i+1}\})\right] \tag{10}$$

where

$$D_{i,k} = \begin{cases} B_k(z_i)\cap B_k(z_{i+1}) & 0 \leq k \leq K-2 \\ B_{K-1}(z_i)\cap B_{K-1}(z_{i+1}) & K-1 \leq k \leq N-2 \end{cases}$$

and

$$A_{i,k} = \begin{cases} 1 & 0 \leq k \leq K-2 \\ \binom{N-\max\left(i+1,\alpha_{x^{(\text{val})}}^{(S,|S|)}\right)}{k-K+1} & K-1 \leq k \leq N-2 \end{cases}$$

Proof. The proof proceeds by standard combinatorial analysis and we refer readers to Appendix E.2. in Jia et al. (2019a) for details. We note that $\alpha_{x^{(\text{val})}}^{(S,|S|)}$ means the index of the farthest data point to $x^{(\text{val})}$ in S. □

Remark 6. *While Jia et al. (2019a) state the above results for soft-label weighted KNN, this algorithm is in fact fairly general and also applies to hard-label weighted KNN.*

Monte Carlo Algorithm. The Monte Carlo approximation algorithm for WKNN-Shapley from Jia et al. (2019a) is a simple adaptation of the famous *permutation sampling* algorithm (Maleki, 2015). We refer the reader to Section 5 of Jia et al. (2019a) for the detailed description and error analysis.

C ADDITIONAL DISCUSSION OF WEIGHTS DISCRETIZATION

Value Rank Preservation. We show that the Shapley value computed based on the discrete weights has the same ranking order compared with the Shapley value computed on the continuous weights (it might create ties but will not reverse the original order).

Here, we consider the binary classification setting since our approach for computing WKNN-Shapley for multi-class classification (in Appendix E) makes a reduction to the binary classification setting. We make a very mild assumption that if $d(x_i, x^{(\mathrm{val})}) \leq d(x_j, x^{(\mathrm{val})})$, then $w_i \geq w_j$. That is, the closer the training point is to the query, the higher the weight the training point will have. This is natural for any reasonable weighting scheme for KNN.

Lemma 16. *Under the assumption of weight function stated above, for any weight assignment function and any two data points z_i, z_j, if $\phi_{z_i}(v) \geq \phi_{z_j}(v)$, then we have $\phi_{z_i}(v^{\texttt{disc}}) \geq \phi_{z_j}(v^{\texttt{disc}})$ where $v^{\texttt{disc}}$ refers to the new utility function after weights discretization.*

Proof. The following cases will result in $\phi_{z_i}(v) \geq \phi_{z_j}(v)$:

Case 1: if $y_i = y^{(\mathrm{val})}$, $y_j \neq y^{(\mathrm{val})}$, it is easy to see that for any $S \subseteq D$ we have $v(S \cup z_i) \geq v(S)$ while $v(S \cup z_j) \leq v(S)$, and hence we have $\phi_{z_i} \geq 0$ while $\phi_{z_i} \leq 0$, regardless of the specific magnitude of the weights. Hence, we still have $\phi_{z_i}(v^{\texttt{disc}}) \geq 0$ and $\phi_{z_j}(v^{\texttt{disc}}) \leq 0$ after weights discretization.

Case 2: if $y_i = y_j = y^{(\mathrm{val})}$, then $\phi_{z_i}(v) \geq \phi_{z_j}(v)$ implies that $d(x_i, x^{(\mathrm{val})}) \leq d(x_j, x^{(\mathrm{val})})$ and $w_i \geq w_j$. This is because for any $S \subseteq D \setminus \{z_i, z_j\}$, if $v(S \cup \{z_j\}) - v(S) = 1$, we must also have $v(S \cup \{z_i\}) - v(S) = 1$. Since weight discretization does not change the rank order of weights w_i and w_j, we still have $\phi_{z_i}(v^{\texttt{disc}}) \geq \phi_{z_j}(v^{\texttt{disc}})$.

Case 3: if $y_i = y_j \neq y^{(\mathrm{val})}$, then $\phi_{z_i}(v) \geq \phi_{z_j}(v)$ implies that $d(x_i, x^{(\mathrm{val})}) \geq d(x_j, x^{(\mathrm{val})})$ and $w_i \leq w_j$. This is because for any $S \subseteq D \setminus \{z_i, z_j\}$, if $v(S \cup \{z_i\}) - v(S) = -1$, we must also have $v(S \cup \{z_j\}) - v(S) = 1$. Since weight discretization does not change the rank order of weights w_i and w_j, we still have $\phi_{z_i}(v^{\texttt{disc}}) \geq \phi_{z_j}(v^{\texttt{disc}})$. □

Value Deviation and Performance on Downstream Tasks. It is difficult to analytically derive or upper bound the deviation of WKNN-Shapley score caused by weight discretization. Therefore, in Appendix G.2, we empirically investigate such value deviation. We also evaluate the impact of weight discretization on WKNN-Shapley's performance on downstream tasks such as mislabeled/noisy data detection. Overall, we conclude that the impact from weight discretization is very small even when the number of discretization bits b is as small as 3.

D ADDITIONAL DETAILS FOR EXACT AND APPROXIMATION ALGORITHM FOR WKNN-SHAPLEY

We state the full version of Theorem 7 here, and defer the proof to Appendix F.

Theorem 17 (Full version of Theorem 7)**.** *When $K > 1$,[3] for $\ell = 1$ we have $\mathtt{F}_i[m, 1, s] = \begin{cases} 1 & s = w_m \\ 0 & s \neq w_m \end{cases}$. We can then compute $\mathtt{F}_i[m, \ell, s]$ for $\ell \geq 2$ with the following relations:*

If $\ell \leq K - 1$, we have

$$\mathtt{F}_i[m, \ell, s] = \sum_{t=1}^{m-1} \mathtt{F}_i[t, \ell - 1, s - w_m] \tag{11}$$

and if $\ell \geq K$, we have

$$\mathtt{F}_i[m, \ell, s] = \begin{cases} 0 & m < i \\ \sum_{t=1}^{m-1} \mathtt{F}_i[t, K-1, s] \binom{N-m}{\ell-K} & m > i \end{cases} \tag{12}$$

Note that we set $\mathtt{F}_i[i, \cdot, \cdot] = 0$ for mathematical convenience.

In the following, we present the pseudocode for our exact computation and approximation algorithm for WKNN-Shapley. For the clarity of presentation, we first show a reader-friendly version but inefficient version of the pseudo-code for the exact computation algorithm from Section 4.1 in Appendix D.1. We then show the pseudo-code that optimizes the runtime (but less readable) in Appendix D.2. We further show the pseudo-code for our deterministic approximation algorithm in Appendix D.3.

Notation. Recall that we use $\mathbf{W}$ to denote the discretized space of $[0, 1]$, where we create 2^b equally spaced points within the interval when we use b bits for discretization. We denote $W := |\mathbf{W}| = 2^b$ the size of the weight space. Furthermore, we use $\mathbf{W}_{(K)}$ to denote the discretized space of $[0, K]$ (where we create $K2^b$ equally spaced points within the interval).

[3] Since Jia et al. (2019a) has shown that weighted KNN-Shapley can be computed in $O(N^K)$ time complexity, we focus on the setting where $K > 1$.

D.1 Reader-friendly Pseudo-code

Here, we show a reader-friendly version of the pseudo-code for our algorithms for computing WKNN-Shapley for binary classification setting. The runtime-optimized version of the pseudo-code is shown in Appendix D.2.

Algorithm 1 Weighted KNN-Shapley for binary classification (reader-friendly version)

Input:

- K – hyperparameter of weighted KNN algorithm.
- $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})})$ – the validation point.
- $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ – sorted training set where $d(x_i, x^{(\mathrm{val})}) \le d(x_j, x^{(\mathrm{val})})$ for any $i \le j$.

Compute the weight w_i for $i \in \{1, \dots, N\}$.
$\widetilde{w}_j = (\mathbf{21}[y^{(\mathrm{val})} = y_j] - 1) w_j$ for $i \in \{1, \dots, N\}$.

for $i \in \{1, \dots, N\}$ **do**

`// Initialize` $\mathtt{F}_i$
Initialize $\mathtt{F}_i[m, \ell, s] = 0$ for $m \in \{1, \dots, N\}, \ell \in \{1, \dots, K-1\}, s \in \mathbf{W}_{(K)}$.
for $m \in \{1, \dots, N\} \setminus \{i\}$ **do**
$\mathtt{F}_i[m, 1, \widetilde{w}_m] = 1$ (Theorem 17)

`// Compute` $\mathtt{F}_i$ (Runtime-optimized version is in Appendix D.2)
for $\ell \in \{2, \dots, K-1\}$ **do**
for $m \in \{\ell, \dots, N\} \setminus \{i\}$ **do**
for $s \in \mathbf{W}_{(K)}$ **do**
$\mathtt{F}_i[m, \ell, s] = \sum_{t=1}^{m-1} \mathtt{F}_i[t, \ell-1, s - \widetilde{w}_m]$ (Equation (11) in Theorem 17)

`// Compute` $\mathtt{R}_{i,m}$ (Runtime-optimized version is in Appendix D.2)
for $m \in \{\max(i+1, K+1), \dots, N\}$ **do**
$$\mathtt{R}_{i,m} = \begin{cases} \sum_{t=1}^{m-1} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \mathtt{F}_i[t, K-1, s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{t=1}^{m-1} \sum_{s \in [-\widetilde{w}_m, -\widetilde{w}_i)} \mathtt{F}_i[t, K-1, s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$$ (Theorem 8)

`// Compute` $\mathtt{G}_{i,\ell}$
$\mathtt{G}_{i,0} = \mathbf{1}[w_i < 0]$.[a]
for $\ell \in \{1, \dots, K-1\}$ **do**
$$\mathtt{G}_{i,\ell} = \begin{cases} \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, 0)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{m \in [N] \setminus i} \sum_{s \in [0, -\widetilde{w}_i)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$$ (Theorem 6)

`// Compute the Shapley value for` z_i
$\phi_{z_i} = \mathtt{sign}(w_i) \left[\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1)}^{N} \frac{\mathtt{R}_{i,m}}{m \binom{m-1}{K}} \right]$.[b] (Theorem 8)

[a] Recall that we define $v(S) = \mathbf{1}\left[\sum_{j=1}^{\min(K,|S|)} \widetilde{w}_{(j)} \ge 0\right]$, hence $v(\{z_i\}) - v(\emptyset) \in \{-1, 0\}$ and is equal to -1 if and only if $w_i < 0$.

[b] $\mathtt{sign}(w) = \begin{cases} 1 & w > 0 \\ 0 & w = 0 \\ -1 & w < 0 \end{cases}$.

D.2 Detailed Pseudo-code used in Implementation

Here, we show the runtime-optimized version of the pseudo-code for our algorithms for computing WKNN-Shapley for binary classification setting. Specifically, the for-loops for computing $\mathtt{F}_i$ and $\mathtt{R}_{i,m}$ can be optimized for efficiency.

Algorithm 2 Weighted KNN-Shapley for binary classification

Input:

- K – hyperparameter of weighted KNN algorithm.
- $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})})$ – the validation point.
- $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ – sorted training set where $d(x_i, x^{(\mathrm{val})}) \leq d(x_j, x^{(\mathrm{val})})$ for any $i \leq j$.

Compute the weight w_i for $i \in \{1, \ldots, N\}$.
$\widetilde{w}_j = (2\mathbf{1}[y^{(\mathrm{val})} = y_j] - 1)w_j$ for $i \in \{1, \ldots, N\}$.

for $i \in \{1, \ldots, N\}$ **do**

`// Initialize` $\mathtt{F}_i$
Initialize $\mathtt{F}_i[m, \ell, s] = 0$ for $m \in \{1, \ldots, N\}, \ell \in \{1, \ldots, K-1\}, s \in \mathbf{W}_{(K)}$.
for $m \in \{1, \ldots, N\} \setminus \{i\}$ **do**
$\mathtt{F}_i[m, 1, \widetilde{w}_m] = 1$ (Theorem 17)

`// Compute` $\mathtt{F}_i$ (Runtime-optimized version)
for $\ell \in \{2, \ldots, K-1\}$ **do**
$F_0[:] = \sum_{t=1}^{\ell-1} \mathtt{F}_i[t, \ell-1, :]$
for $m \in \{\ell, \ldots, N\} \setminus \{i\}$ **do**
for $s \in \mathbf{W}_{(K)}$ **do**
$\mathtt{F}_i[m, \ell, s] = F_0[s - w_m]$

`// Compute` $\mathtt{R}_{i,m}$ (Runtime-optimized version)
for $s \in \mathbf{W}_{(K)}$ **do**
$R_0[s] = \sum_{t=1, t\neq i}^{\max(i+1, K+1)-1} \mathtt{F}_i[t, K-1, s]$.
for $m \in \{\max(i+1, K+1), \ldots, N\}$ **do**
$$\mathtt{R}_{i,m} = \begin{cases} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} R_0[s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{s \in [-\widetilde{w}_m, -\widetilde{w}_i)} R_0[s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$$
$R_0 = R_0 + \mathtt{F}_i[m, K-1, :]$

`// Compute` $\mathtt{G}_{i,\ell}$
$\mathtt{G}_{i,0} = \mathbf{1}[w_i < 0]$.[a]
for $\ell \in \{1, \ldots, K-1\}$ **do**
$$\mathtt{G}_{i,\ell} = \begin{cases} \sum_{m \in [N]\setminus i} \sum_{s \in [-\widetilde{w}_i, 0)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{m \in [N]\setminus i} \sum_{s \in [0, -\widetilde{w}_i)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$$ (Theorem 6)

`// Compute the Shapley value for` z_i
$\phi_{z_i} = \mathtt{sign}(w_i) \left[\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1)}^{N} \frac{\mathtt{R}_{i,m}}{m\binom{m-1}{K}} \right]$.[b] (Theorem 8)

[a] Recall that we define $v(S) = \mathbf{1}\left[\sum_{j=1}^{\min(K,|S|)} \widetilde{w}_{(j)} \geq 0\right]$, hence $v(\{z_i\}) - v(\emptyset) \in \{-1, 0\}$ and is equal to -1 if and only if $w_i < 0$.

[b] $\mathtt{sign}(w) = \begin{cases} 1 & w > 0 \\ 0 & w = 0 \\ -1 & w < 0 \end{cases}$.

D.3 Detailed Pseudo-code for approximation algorithm

Here, we show the pseudo-code for our deterministic approximation algorithms for WKNN-Shapley from Section 4.2. We highlight its difference with the exact computation algorithm.

Algorithm 3 Approximation of Weighted KNN-Shapley for binary classification

Input:

- K – hyperparameter of weighted KNN algorithm.
- $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})})$ – the validation point.
- $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ – sorted training set where $d(x_i, x^{(\mathrm{val})}) \le d(x_j, x^{(\mathrm{val})})$ for any $i \le j$.
- $M^\star$ – hyperparameter for SV approximation (Section 4.2). $M^\star = N$ for exact SV calculation.

Compute the weight $w_i = \omega_{x^{(\mathrm{val})}}(x_i)$ for $i \in \{1, \dots, N\}$.
$\widetilde{w}_j = (2\mathbf{1}[y^{(\mathrm{val})} = y_j] - 1) w_j$ for $i \in \{1, \dots, N\}$.

for $i \in \{1, \dots, N\}$ **do**

  `// Initialize` $\mathtt{F}_i$
  Initialize $\mathtt{F}_i[m, \ell, s] = 0$ for $m \in \{1, \dots, M^\star\}, \ell \in \{1, \dots, K-1\}, s \in \mathbf{W}_{(K)}$.
  for $m \in \{1, \dots, M^\star\} \setminus \{i\}$ **do**
    $\mathtt{F}_i[m, 1, \widetilde{w}_m] = 1$ (Theorem 17)

  `// Compute` $\mathtt{F}_i$ (Runtime-optimized version)
  for $\ell \in \{2, \dots, K-1\}$ **do**
    $F_0[:] = \sum_{t=1}^{\ell-1} \mathtt{F}_i[t, \ell-1, :]$
    for $m \in \{\ell, \dots, M^\star\} \setminus \{i\}$ **do**
      for $s \in \mathbf{W}_{(K)}$ **do**
        $\mathtt{F}_i[m, \ell, s] = F_0[s - w_m]$

  `// Compute` $\mathtt{R}_{i,m}$ (Runtime-optimized version)
  for $s \in \mathbf{W}_{(K)}$ **do**
    $R_0[s] = \sum_{t=1, t\neq i}^{\max(i+1, K+1)-1} \mathtt{F}_i[t, K-1, s]$.
  for $m \in \{\max(i+1, K+1), \dots, M^\star\}$ **do**
    $\mathtt{R}_{i,m} = \begin{cases} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} R_0[s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{s \in [-\widetilde{w}_m, -\widetilde{w}_i)} R_0[s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$
    $R_0 = R_0 + \mathtt{F}_i[m, K-1, :]$

  `// Compute` $\mathtt{G}_{i,\ell}$
  $\widetilde{\mathtt{G}}_{i,0}^{(M^\star)} = \mathbf{1}[w_i < 0]$.[a]
  for $\ell \in \{1, \dots, K-1\}$ **do**
    $\widetilde{\mathtt{G}}_{i,\ell}^{(M^\star)} = \begin{cases} \sum_{m \in [M^\star] \setminus i} \sum_{s \in [-\widetilde{w}_i, 0)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{m \in [M^\star] \setminus i} \sum_{s \in [0, -\widetilde{w}_i)} \mathtt{F}_i[m, \ell, s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases}$ (Definition 10)

  `// Compute the Shapley value for` z_i
  $\widehat{\phi}_{z_i}^{(M^\star)} = \mathtt{sign}(w_i) \left[\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\widetilde{\mathtt{G}}_{i,\ell}^{(M^\star)}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1, K+1)}^{M^\star} \frac{\mathtt{R}_{i,m}}{m \binom{m-1}{K}} \right]$.[b] (Definition 10)

[a] Recall that we define $v(S) = \mathbf{1}\left[\sum_{j=1}^{\min(K, |S|)} \widetilde{w}_{(j)} \ge 0\right]$, hence $v(\{z_i\}) - v(\emptyset) \in \{-1, 0\}$ and is equal to -1 if and only if $w_i < 0$.

[b] $\mathtt{sign}(w) = \begin{cases} 1 & w > 0 \\ 0 & w = 0 \\ -1 & w < 0 \end{cases}$.

D.4 Expanded Discussion for the Selection of $M^\star$ for Deterministic Approximation Algorithm

Determining a universally applicable heuristic for setting $M^\star$ is challenging. As we mentioned in Remark 5, ideally, we would like to pick the smallest $M^\star$ such that the error bound $\varepsilon(M^\star)$ from Theorem 12 is significantly smaller than $|\phi_{z_i}|$ for a significant portion of z_is. However, determining a universally applicable heuristic for setting $M^\star$ is challenging due to the varying magnitude of ϕ_{z_i} across different datasets, which are difficult to anticipate. For example, in the case where all but one data point are "null players", that single data point will possess a value of $v(D)$, while all others will be valued at 0. On the other hand, if all data points are identical, each will receive a value of $v(D)/N$.

We can select $M^\star$ in an adaptive way (not used in our experiment). As the magnitude of WKNN-Shapley can vary significantly depending on the specific dataset, it is more reasonable to select $M^\star$ in an adaptive way. Specifically, we can compute $(\widehat{\phi}_{z_i}^{(M^\star)})_{i\in[N]}$ for each of $M^\star = K+1, \ldots,$. We can keep increment $M^\star$ until the magnitude of $\varepsilon(M^\star)$ is substantially smaller than (e.g., $< 10\%$) the magnitude of $\widehat{\phi}_{z_i}^{(M^\star)}$ for a majority of z_is. We can easily modify Algorithm 3 so that this approach does not increase the overall runtime since $\widehat{\phi}_{z_i}^{(M^\star)}$ can be easily computed from $\widehat{\phi}_{z_i}^{(M^\star-1)}$ and the additionally computed $\mathtt{F}_i[M^\star, \cdot, \cdot]$. That is, from Definition 10, we can easily see that when $y_i = y^{(\mathrm{val})}$,

$$\widehat{\phi}_{z_i}^{(M^\star)} = \widehat{\phi}_{z_i}^{(M^\star-1)} + \left(\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\sum_{s\in[-\widetilde{w}_i,0)} \mathtt{F}_i\left[M^\star, \ell, s\right]}{\binom{N-1}{\ell}} + \frac{\mathtt{R}_{i,M^\star}}{M^\star \binom{M^\star-1}{K}} \right) \tag{13}$$

and when $y_i \neq y^{(\mathrm{val})}$,

$$\widehat{\phi}_{z_i}^{(M^\star)} = \widehat{\phi}_{z_i}^{(M^\star-1)} - \left(\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\sum_{s\in[0,-\widetilde{w}_i)} \mathtt{F}_i\left[M^\star, \ell, s\right]}{\binom{N-1}{\ell}} + \frac{\mathtt{R}_{i,M^\star}}{M^\star \binom{M^\star-1}{K}} \right) \tag{14}$$

In our experiment, we find $M^\star = \sqrt{N}$ works well across all benchmark datasets (ablation study in Appendix G.5). However, in our experiment, we find that the performance of approximated WKNN-Shapley on the downstream tasks such as mislabeled and noisy data detection, are relatively stable across a wide range of choices of $M^\star$ (see Appendix G.5). This is likely because of the nice property of our approximated WKNN-Shapley in preserving the fairness properties of the exact WKNN-Shapley (Theorem 13). Hence, in our experiment, we do not follow the adaptive process of selecting $M^\star$, but adopt the simple rule of $M^\star = \sqrt{N}$, which already works very well.

E EXTENSION TO MULTI-CLASS CLASSIFICATION SETTING

In Section 4, we have developed efficient solutions for computing and approximating the Data Shapley of weighted hard-label KNN *binary* classifiers. Directly adapting the same techniques from the binary to multi-class classification setting, however, can significantly increase the overall time complexity (see Appendix E.1 for details). Hence, in Appendix E.2, we introduce an alternative utility function to measure the performance of WKNN classifiers. This new utility function provides more detailed confidence information about the KNN classifier, as opposed to the original utility function in (2), which only provides basic zero-one correctness. More importantly, computing the Data Shapley in terms of the new utility function can be conveniently reduced to the WKNN-Shapley computation for binary classifiers without significantly increasing time complexity, a benefit leveraged from the linearity axiom of the Shapley value.

Notation Review. Recall that we use $\mathbf{W}$ to denote the discretized space of $[0,1]$, where we create 2^b equally spaced points within the interval when we use b bits for discretization. We denote $W := |\mathbf{W}| = 2^b$ the size of the weight space. Furthermore, we use $\mathbf{W}_{(K)}$ to denote the discretized space of $[0,K]$ (where we create $K2^b$ equally spaced points within the interval). We use $\texttt{NB}_{x^{(\text{val})},K}(S)$ to denote the set of data points that is within the K-nearest neighbors of $x^{(\text{val})}$ among S. We use $\alpha^{(S,j)}_{x^{(\text{val})}}$ denotes the index (among D) of jth closest data point in S to $x^{(\text{val})}$.

E.1 Direct Extension from Binary Classification Setting can be Inefficient

We first discuss a simple, direct extension of our exact WKNN-Shapley algorithm from binary to multi-class classification setting. In Algorithm 1, the main idea is to maintain a record of $\texttt{F}_i[m,\ell,s]$ for a singular scalar value s which represents the summation of "signed weights" $\widetilde{w}_j$. In order to extend this approach to the multi-class setting, it is natural to enhance this scalar representation to a "histogram" depiction, $\texttt{F}_i[m,\ell,\mathbf{s}]$, where $\mathbf{s}$ is the vector sum of weights for each data point, and the weights are in the form of one-hot encoding. That is, in the multi-class setting, $\texttt{F}_i$ is augmented to record the number of subsets such that the sum of weights of the data points in the one-hot encoding is equal to the histogram $\mathbf{s}$ (subject to the conditions analog to those in Definition 5). Denote $\mathbf{e}_y \in [C]$ the one-hot encoding of the label, with 1 on yth entry and 0 otherwise. The augmented $\texttt{F}_i$ is defined as follows:

Definition 18. *Let $\texttt{F}_i[m,\ell,\mathbf{s}]$ denote the count of subsets $S \subseteq D \setminus \{z_i\}$ of size ℓ that satisfy the conditions below: **(1)** $\texttt{Cond}_{KNN}$, **(2)** Within S, the data point x_m is the $\min(\ell,K)$-th closest to the query example $x^{(\text{val})}$, and **(3)** $\sum_{j=1}^{\min(\ell,K-1)} w_{\alpha^{(S,j)}_{x^{(\text{val})}}} \mathbf{e}_{y_{\alpha^{(S,j)}_{x^{(\text{val})}}}} = \mathbf{s}$.*

The results in the maintext (Theorem 6, 7, and 8) can be easily adapted to this more generalized definition of $\texttt{F}_i$. While this direct extension can compute the exact Data Shapley for the utility function in (2), it has a time complexity of $O(K^{1+C}N^2W^C)$ as we need to record $\texttt{F}_i[m,\ell,\mathbf{s}]$ for all possible histograms $\mathbf{s} \in \mathbf{W}^C_{(K)}$, where $\mathbf{W}^C_{(K)}$ denotes the product space of $\mathbf{W}_{(K)}$. This is manageable for datasets with a modest size of class space. However, for datasets with a large class space, this complexity can render the runtime prohibitively large.

E.2 Alternative Utility Function that Enables More Efficient Computation of WKNN-Shapley

Due to the above-mentioned computational bottleneck, we introduce an alternative utility function for weighted KNN classifiers, which not only reflects the KNN classifiers' performance but also paves the way for a more efficient Data Shapley computation analogous to that of the binary setting. This is another instantiation of the rationale of developing advanced data valuation techniques with proper adjustment to the utility function.

Alternative Utility Function for Weighted Hard-Label KNN Classifiers. For a class $c \in [C] \setminus \{y^{(\text{val})}\}$, we denote

$$v^{(c)}(S; z^{(\text{val})}) := \mathbf{1}\left[\sum_{j=1}^{\min(K,|S^{(c)}|)} w_{\alpha^{(S^{(c)},j)}_{x^{(\text{val})}}} \mathbf{1}\left[y_{\alpha^{(S^{(c)},j)}_{x^{(\text{val})}}} = y^{(\text{val})}\right] \geq \sum_{j=1}^{\min(K,|S^{(c)}|)} w_{\alpha^{(S^{(c)},j)}_{x^{(\text{val})}}} \mathbf{1}\left[y_{\alpha^{(S^{(c)},j)}_{x^{(\text{val})}}} = c\right]\right] \tag{15}$$

where $S^{(c)} := \{(x,y) \in S : y \in \{y^{(\text{val})}, c\}\}$ is the subset of S whose labels are either $y^{(\text{val})}$ and c. We propose an

alternative utility function as follows:

$$\widetilde{v}(S; z^{(\mathrm{val})}) := \frac{1}{C-1} \sum_{c \in [C] \setminus y^{(\mathrm{val})}} v^{(c)}(S^{(c)}; z^{(\mathrm{val})}) \tag{16}$$

Note that for binary classifiers, the new utility function $\widetilde{v}$ reduces to the original v. **Interpretation of the Alternative Utility Function:** The alternative utility function, $\widetilde{v}$, captures a more fine-grained view of the classifier's performance. Instead of just deciding based on whether a prediction is correct as the original utility function in (2), it reduces the multi-class classification game as multiple binary-class classification games, and assesses how many times the correct class, $y^{(\mathrm{val})}$, are correctly being predicted in those subgames. Hence, $\widetilde{v}$ provides insight into not just the correctness, but also the relative confidence of a prediction with respect to other classes.

Data Shapley for $\widetilde{v}$. The linearity axiom of the Shapley value provides that

$$\phi_{z_i}(\widetilde{v}) = \frac{1}{C-1} \sum_{c \in [C] \setminus y^{(\mathrm{val})}} \phi_{z_i}(v^{(c)})$$

Denote the subset $D_{y^{(\mathrm{val})},c} \subseteq D$ such that $D_{y^{(\mathrm{val})},c} := \{(x, y) \in D : y \in \{y^{(\mathrm{val})}, c\}\}$. Observe that $\phi_{z_i}(v^{(c)}) = 0$ for all $z_i \notin D_{y^{(\mathrm{val})},c}$, and $\phi_{z_i}(v^{(c)})$ for $z_i \in D_{y^{(\mathrm{val})},c}$ can be easily computed with WKNN-Shapley for binary classification setting (Algorithm 2). Hence, we can first compute $\phi_{z_i}(v^{(c)})$ for each $c \in [C] \setminus y^{(\mathrm{val})}$ individually, and then aggregate these values.

Algorithm 4 Weighted KNN-Shapley for multi-class classification

1: **Input:**
- K – hyperparameter of weighted KNN algorithm.
- $z^{(\mathrm{val})} = (x^{(\mathrm{val})}, y^{(\mathrm{val})})$ – the validation point.
- $D = \{z_i = (x_i, y_i)\}_{i=1}^N$ – sorted training set where $d(x_i, x^{(\mathrm{val})}) \leq d(x_j, x^{(\mathrm{val})})$ for any $i \leq j$.
- C – number of classes

2: **for** $c \in [C] \setminus \{y^{(\mathrm{val})}\}$ **do**
3: $\quad \phi_{z_i}(v^{(c)}) = 0$ for $z_i \notin D_{y^{(\mathrm{val})},c}$.
4: $\quad$ Compute $\phi_{z_i}(v^{(c)})$ for $z_i \in D_{y^{(\mathrm{val})},c}$ by executing Algorithm 2 on $D_{y^{(\mathrm{val})},c}$.
5: **Return:** $\phi_{z_i}(\widetilde{v}) = \frac{1}{C-1} \sum_{c \in [C] \setminus y^{(\mathrm{val})}} \phi_{z_i}(v^{(c)})$ for $z_i \in D$.

Remark 7. *We can also replace the exact computation of $\phi_{z_i}(v^{(c)})$ in line 4 by our deterministic approximation algorithm (Algorithm 3) and obtain a deterministic approximation for $\phi_{z_i}(\widetilde{v})$. The error bound is simply the average of error bounds for computing $\phi_{z_i}(v^{(c)})$ for each of $c \in [C] \setminus \{y^{(\mathrm{val})}\}$.*

While this might imply an inevitable factor of C in the computational complexity, note that the "effective dataset" for $v^{(c)}$ is the subset $D_{y^{(\mathrm{val})},c} \subseteq D$ that comprises only data points labeled $y^{(\mathrm{val})}$ or c. As a result, the computational time to compute the Shapley value for $v^{(c)}$ reduces to $O(K^2 |D_{y^{(\mathrm{val})},c}|^2 W)$. This provides a huge runtime saving when the dataset is balanced.

Theorem 19. *For a class-balanced training dataset D with C classes, computing the exact WKNN-Shapley $\{\phi_{z_i}(\widetilde{v})\}_{z_i \in D}$ achieves runtime $O\left(\frac{K^2 N^2 W}{C}\right)$.*

Proof. When the dataset D is balanced, we have $|D_{y^{(\mathrm{val})},c}| = \frac{2N}{C}$. Hence, the runtime of computing $\{\phi_{z_i}(v^{(c)})\}_{z_i \in D}$ is $O\left(\frac{K^2 N^2 W}{C^2}\right)$, and hence the total runtime is $O\left(\frac{K^2 N^2 W}{C}\right)$. □

Remarkably, this methodology is even more efficient than its binary classification counterpart.

F MISSING PROOFS

Notation Review. Recall that we use $\mathbf{W}$ to denote the discretized space of $[0,1]$, where we create 2^b equally spaced points within the interval when we use b bits for discretization. We denote $W := |\mathbf{W}| = 2^b$ the size of the weight space. Furthermore, we use $\mathbf{W}_{(K)}$ to denote the discretized space of $[0,K]$ (where we create $K2^b$ equally spaced points within the interval). We use $\texttt{NB}_{x^{(\text{val})},K}(S)$ to denote the set of data points that is within the K-nearest neighbors of $x^{(\text{val})}$ among S. We use $\alpha^{(S,j)}_{x^{(\text{val})}}$ denotes the index (among D) of jth closest data point in S to $x^{(\text{val})}$.

Theorem 20 (Restate of Theorem 2)**.** *For any data point $z_i \in D$ and any subset $S \subseteq D \setminus \{z_i\}$, the marginal contribution has the expression as follows:*

$$v(S\cup\{z_i\}) - v(S) = \begin{cases} 1 & \text{if } y_i = y^{(\text{val})}, \texttt{Cond}_{KNN}, \texttt{Cond}_{0to1} \\ -1 & \text{if } y_i \neq y^{(\text{val})}, \texttt{Cond}_{KNN}, \texttt{Cond}_{1to0} \\ 0 & \text{Otherwise} \end{cases} \tag{17}$$

where

$$\texttt{Cond}_{KNN} := z_i \text{ is within } K \text{ nearest neighbors of } x^{(\text{val})} \text{ among } S\cup\{z_i\} \tag{18}$$

$$\texttt{Cond}_{0to1} := \begin{cases} \sum_{z_j\in S} \widetilde{w}_j \in [-\widetilde{w}_i, 0) & \text{if } |S| \leq K-1 \\ \sum_{j=1}^{K-1} \widetilde{w}_{\alpha^{(S,j)}_{x^{(\text{val})}}} \in \left[-w_i, -\widetilde{w}_{\alpha^{(S,K)}_{x^{(\text{val})}}}\right) & \text{if } |S| \geq K \end{cases} \tag{19}$$

$$\texttt{Cond}_{1to0} := \begin{cases} \sum_{z_j\in S} \widetilde{w}_j \in [0, -\widetilde{w}_i) & \text{if } |S| \leq K-1 \\ \sum_{j=1}^{K-1} \widetilde{w}_{\alpha^{(S,j)}_{x^{(\text{val})}}} \in \left[-\widetilde{w}_{\alpha^{(S,K)}_{x^{(\text{val})}}}, -w_i\right) & \text{if } |S| \geq K \end{cases} \tag{20}$$

Proof. First of all, we observe that if $z_i \notin \texttt{NB}_{x^{(\text{val})},K}(S\cup\{z_i\})$, i.e., if z_i is not within the K nearest neighbors of the queried example $x^{(\text{val})}$ among the subset $S\cup\{z_i\}$, then the prediction of KNN classifier does not change, and hence we know that $v(S\cup\{z_i\}) = v(S)$. Hence $\texttt{Cond}_{KNN}$ is necessary for $v(S\cup\{z_i\}) - v(S)$ to be non-zero.

If $z_i \in \texttt{NB}_{x^{(\text{val})},K}(S\cup\{z_i\})$, we divide into two cases: ① If $|S| \leq K-1$ we know that adding z_i will not exclude any other data point from the K nearest neighbors of $x^{(\text{val})}$. Hence $v(S\cup\{z_i\}) - v(S) = 1$ only if $y_i = y^{(\text{val})}$ and $\sum_{z_j\in S}\widetilde{w}_j \in [-\widetilde{w}_i, 0)$, and $v(S\cup\{z_i\}) - v(S) = -1$ only if $y_i \neq y^{(\text{val})}$ and $\sum_{z_j\in S}\widetilde{w}_j \in [0, -\widetilde{w}_i)$. ② If $|S| \geq K$ we know that adding z_i will exclude the original Kth nearest neighbors of $x^{(\text{val})}$ among dataset S. Hence, $v(S\cup\{z_i\}) - v(S) = 1$ only if $y_i = y^{(\text{val})}$ and $\sum_{j=1}^{K-1}\widetilde{w}_{\alpha_{x^{(\text{val})}}(S,j)} \in \left[-w_i, -\widetilde{w}_{\alpha_{x^{(\text{val})}}(S,K)}\right)$, and $v(S\cup\{z_i\}) - v(S) = -1$ only if $y_i \neq y^{(\text{val})}$ and $\sum_{j=1}^{K-1}\widetilde{w}_{\alpha_{x^{(\text{val})}}(S,j)} \in \left[-\widetilde{w}_{\alpha_{x^{(\text{val})}}(S,K)}, -w_i\right)$. □

Theorem 21 (Full version of Theorem 7)**.** *When $K > 1$,[4] for $\ell = 1$ we have $\texttt{F}_i[m,1,s] = \begin{cases} 1 & s = w_m \\ 0 & s \neq w_m \end{cases}$. We can then compute $\texttt{F}_i[m,\ell,s]$ for $\ell \geq 2$ with the following relations:*

If $\ell \leq K-1$, we have

$$\texttt{F}_i[m,\ell,s] = \sum_{t=1}^{m-1} \texttt{F}_i[t,\ell-1,s-w_m] \tag{21}$$

and if $\ell \geq K$, we have

$$\texttt{F}_i[m,\ell,s] = \begin{cases} 0 & m < i \\ \sum_{t=1}^{m-1} \texttt{F}_i[t,K-1,s]\binom{N-m}{\ell-K} & m > i \end{cases} \tag{22}$$

Note that we set $\texttt{F}_i[i,\cdot,\cdot] = 0$ for mathematical convenience.

[4] Since Jia et al. (2019a) has shown that weighted KNN-Shapley can be computed in $O(N^K)$ time complexity, we focus on the setting where $K > 1$.

Proof. **Base case of $\ell = 1$:** by definition of $\mathtt{F}_i$, if $\ell = 1$ the dataset that satisfy the conditions required for $\mathtt{F}_i[m, 1, \cdot]$ can only be the singleton $\{z_m\}$, and hence the base case is straightforward.

Case of $\ell \leq K-1$: the inclusion of z_i in $\mathtt{NB}_{x^{(\mathrm{val})},K}(S \cup \{z_i\})$ is guaranteed for this range of ℓ. The datasets that satisfy the conditions required for $\mathtt{F}_i[m, \ell, s]$ can be partitioned based on the $(\ell-1)$th closest point to $x^{(\mathrm{val})}$, which leads to the recursive relation in (21).

Case of $\ell \geq K$: Since x_m is the K-th nearest data point to $x^{(\mathrm{val})}$ within S, we have $\mathtt{F}_i[m, \ell, s] = 0$ for any $m < i$ since z_i is also required to be within K nearest neighbor to $x^{(\mathrm{val})}$. When $m > i$, the datasets that satisfy the conditions required for $\mathtt{F}_i[m, \ell, s]$ can be partitioned based on the $(K-1)$th closest point to $x^{(\mathrm{val})}$, which leads to the relation in (22) by simple combinatorial analysis. □

Theorem 22 (Restate of Theorem 8)**.** *For a weighted, hard-label KNN binary classifier using the utility function given by (3), the Shapley value of data point z_i can be expressed as:*

$$\phi_{z_i} = \mathit{sign}(w_i) \left[\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1)}^{N} \frac{\mathtt{R}_{i,m}}{m\binom{m-1}{K}} \right] \tag{23}$$

where

$$\mathtt{R}_{i,m} := \begin{cases} \sum_{t=1}^{m-1} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \mathtt{F}_i[t, K-1, s] & \text{for } y_i = y^{(\mathrm{val})} \\ \sum_{t=1}^{m-1} \sum_{s \in [-\widetilde{w}_m, -\widetilde{w}_i)} \mathtt{F}_i[t, K-1, s] & \text{for } y_i \neq y^{(\mathrm{val})} \end{cases} \tag{24}$$

Proof. We state the proof for the case where $y_i = y^{(\mathrm{val})}$, and the proof for the case where $y_i \neq y^{(\mathrm{val})}$ is nearly identical. Recall that

$$\mathtt{G}_{i,\ell} = \begin{cases} \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, 0)} \mathtt{F}_i[m, \ell, s] & \ell \leq K-1 \\ \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \mathtt{F}_i[m, \ell, s] & \ell \geq K \end{cases}$$

if $y_i = y^{(\mathrm{val})}$.

When $\ell \geq K$, we have

$$\begin{aligned} \mathtt{G}_{i,\ell} &= \sum_{m \in [N] \setminus i} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \mathtt{F}_i[m, \ell, s] \\ &= \sum_{m=\max(i+1,K+1)}^{N} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \mathtt{F}_i[m, \ell, s] \\ &= \sum_{m=\max(i+1,K+1)}^{N} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \binom{N-m}{\ell-K} \sum_{t=1, t \neq i}^{m-1} \mathtt{F}_i[t, K-1, s] \\ &= \sum_{m=\max(i+1,K+1)}^{N} \binom{N-m}{\ell-K} \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \sum_{t=1, t \neq i}^{m-1} \mathtt{F}_i[t, K-1, s] \\ &= \sum_{m=\max(i+1,K+1)}^{N} \binom{N-m}{\ell-K} \mathtt{R}_{i,m} \end{aligned}$$

where $\mathtt{R}_{i,m} = \sum_{s \in [-\widetilde{w}_i, -\widetilde{w}_m)} \sum_{t=1, t \neq i}^{m-1} \mathtt{F}_i[t, K-1, s]$.

$$\begin{aligned}
\sum_{\ell=K}^{N-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}} &= \sum_{\ell=K}^{N-1} \sum_{m=\max(i+1,K+1)}^{N} \frac{\binom{N-m}{\ell-K}\mathtt{R}_{i,m}}{\binom{N-1}{\ell}} \\
&= \sum_{m=\max(i+1,K+1)}^{N} \mathtt{R}_{i,m} \sum_{\ell=K}^{N-1} \frac{\binom{N-m}{\ell-K}}{\binom{N-1}{\ell}} \\
&= \sum_{m=\max(i+1,K+1)}^{N} \mathtt{R}_{i,m} \left(\sum_{\ell=K}^{N-1} \frac{\binom{m-1}{K}\binom{N-m}{\ell-K}}{\binom{N-1}{\ell}} \right) \binom{m-1}{K}^{-1} \\
&= \sum_{m=\max(i+1,K+1)}^{N} \mathtt{R}_{i,m} \left(\frac{N}{m} \right) \binom{m-1}{K}^{-1}
\end{aligned}$$

□

Theorem 23 (Restate of Theorem 9)**.** *Algorithm 2 (in Appendix D.2) computes the exact Shapley value and achieves $O(K^2N^2W)$ time complexity.*

Proof. It is easy to see that the for-loop for computing $\mathtt{F}_i$ for $\ell \le K$ requires a runtime of $O(KN|\mathbf{W}_{(K)}|)$. The for-loop for computing $\mathtt{R}_{i,m}$ requires a runtime of $O(N|\mathbf{W}_{(K)}|)$. The for-loop for computing $\mathtt{G}_{i,\ell}$ for $\ell \le K$ requires a runtime of $O(KN|\mathbf{W}_{(K)}|)$. All of these subroutines are included in the outside for-loop for computing ϕ_{z_i} for all $z_i \in D$. Hence, the overall runtime is $O(KN^2|\mathbf{W}_{(K)}|) = O(K^2N^2W)$. □

Theorem 24 (Restate of Theorem 11)**.** *Algorithm 3 (in Appendix D.3) computes the approximated WKNN-Shapley $\widehat{\phi}_{z_i}^{(M^\star)}$ for all $z_i \in D$ and achieves a total runtime of $O(WK^2NM^\star)$.*

Proof. It is easy to see that the for-loop for computing $\mathtt{F}_i$ for $\ell \le K$ requires a runtime of $O(KM^\star|\mathbf{W}_{(K)}|)$. The for-loop for computing $\mathtt{R}_{i,m}$ requires a runtime of $O(M^\star|\mathbf{W}_{(K)}|)$. The for-loop for computing $\mathtt{G}_{i,\ell}$ for $\ell \le K$ requires a runtime of $O(KM^\star|\mathbf{W}_{(K)}|)$. All of these subroutines are included in the outside for-loop for computing ϕ_{z_i} for all $z_i \in D$. Hence, the overall runtime is $O(KNW|\mathbf{W}_{(K)}|) = O(WK^2NM^\star)$. □

Theorem 25 (Restate of Theorem 12)**.** *For any $z_i \in D$, the approximated Shapley value $\widehat{\phi}_{z_i}^{(M^\star)}$ **(1)** shares the same sign as ϕ_{z_i}, **(2)** ensures $\left|\widehat{\phi}_{z_i}^{(M^\star)}\right| \le |\phi_{z_i}|$, and **(3)** has the approximation error bounded by $\left|\widehat{\phi}_{z_i}^{(M^\star)} - \phi_{z_i}\right| \le \varepsilon(M^\star)$ where*

$$\varepsilon(M^\star) := \sum_{m=M^\star+1}^{N} \left(\frac{1}{m-K} - \frac{1}{m} \right) + \sum_{\ell=1}^{K-1} \frac{\binom{N}{\ell} - \binom{M^\star}{\ell}}{N\binom{N-1}{\ell}} = O\left(K/M^\star\right)$$

Proof. The property **(1)** follows from that the sign of both exact and approximated WKNN-Shapley only depends on whether $y_i = y^{(\mathrm{val})}$. The property **(2)** follows from the approximation algorithm only counts *part* of the subproblems, and hence $\left|\widehat{\phi}_{z_i}^{(M^\star)}\right| \le |\phi_{z_i}|$. We now prove property **(3)**.

In the exact algorithm 1, we have

$$\phi_i = \underbrace{\frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\mathtt{G}_{i,\ell}}{\binom{N-1}{\ell}}}_{(A)} + \underbrace{\sum_{m=\max(i+1,K+1)}^{N} \mathtt{R}_{i,m} \left(\frac{1}{m} \right) \binom{m-1}{K}^{-1}}_{(B)}$$

First of all, note that

$$\sum_{s \in \mathbf{W}} \mathtt{F}_i\left[m, \ell, s\right] = \binom{m-1-\mathbf{1}[i<m]}{\ell-1} \le \binom{m-1}{\ell-1}$$

for any $\ell \leq K$ since $\sum_{s\in\mathbf{W}} \mathtt{F}_i\left[m,\ell,s\right]$ is essentially the total number of subsets $S \subseteq D \setminus z_i$ of size ℓ where z_m is the farthest data point to the query example $x^{(\mathrm{val})}$.

Now, denote

$$\widetilde{\mathtt{G}}_{i,\ell} := \sum_{m=1}^{M^\star} \sum_{s\in[-\widetilde{w}_i,0)} \mathtt{F}_i\left[m,\ell,s\right]$$

for $1 \leq \ell \leq K-1$. The gap between $\mathtt{G}_{i,\ell}$ and $\widetilde{\mathtt{G}}_{i,\ell}$ can be bounded as follows:

$$\begin{aligned}
\left|\widetilde{\mathtt{G}}_{i,\ell} - \mathtt{G}_{i,\ell}\right| &= \sum_{m=M^\star+1}^{N} \sum_{s\in[-\widetilde{w}_i,0)} \mathtt{F}_i\left[m,\ell,s\right] \\
&\leq \sum_{m=M^\star+1}^{N} \sum_{s\in\mathbf{W}} \mathtt{F}_i\left[m,\ell,s\right] \\
&\leq \sum_{m=M^\star+1}^{N} \binom{m-1}{\ell-1} \\
&= \sum_{m=\ell}^{N} \binom{m-1}{\ell-1} - \sum_{m=\ell}^{M^\star} \binom{m-1}{\ell-1} \\
&= \binom{N}{\ell} - \binom{M^\star}{\ell}
\end{aligned}$$

Now we bound the error from taking the approximation $\widehat{\mathtt{R}}_{i,m} = 0$ for $m \geq M^\star + 1$. Since we have

$$\begin{aligned}
\mathtt{R}_{i,m} &= \sum_{t=1}^{m-1} \sum_{s\in[-\widetilde{w}_i,-\widetilde{w}_m)} \mathtt{F}_i[t,K-1,s] \\
&\leq \sum_{t=1}^{m-1} \binom{t-1}{K-2} \\
&= \binom{m-1}{K-1}
\end{aligned}$$

Hence

$$\begin{aligned}
\sum_{m=\max(i+1,K+1,M^\star+1)}^{N} \mathtt{R}_{i,m} \binom{1}{m} \binom{m-1}{K}^{-1} &\leq \sum_{m=\max(i+1,K+1,M^\star+1)}^{N} \binom{m-1}{K-1} \binom{1}{m} \binom{m-1}{K}^{-1} \\
&\leq \sum_{m=M^\star+1}^{N} \binom{m-1}{K-1} \binom{1}{m} \binom{m-1}{K}^{-1} \\
&= \sum_{m=M^\star+1}^{N} \frac{K}{m(m-K)} \\
&= \sum_{m=M^\star+1}^{N} \left(\frac{1}{m-K} - \frac{1}{m}\right)
\end{aligned}$$

Hence, for any data point z_i, we have

$$\begin{aligned}
\left|\widehat{\phi}_{z_i}^{(M^\star)} - \phi_i\right| &= \frac{1}{N} \sum_{\ell=0}^{K-1} \frac{\left|\mathtt{G}_{i,\ell} - \widetilde{\mathtt{G}}_{i,\ell}\right|}{\binom{N-1}{\ell}} + \sum_{m=\max(i+1,K+1,M^\star+1)}^{N} \mathtt{R}_{i,m} \binom{1}{m} \binom{m-1}{K}^{-1} \\
&\leq \frac{1}{N} \sum_{\ell=1}^{K-1} \frac{\binom{N}{\ell} - \binom{M^\star}{\ell}}{\binom{N-1}{\ell}} + \sum_{m=M^\star+1}^{N} \left(\frac{1}{m-K} - \frac{1}{m}\right)
\end{aligned}$$

□

Theorem 26 (Restate of Theorem 13)**.** *The approximated Shapley value* $\{\widehat{\phi}_{z_i}^{(M^\star)}\}_{z_i \in D}$ *satisfies symmetry and null player axiom.*

Proof. **Null Player Axiom.** If a data point z_i is a null player (i.e., $v(S \cup z_i) = v(S)$ for all $S \subseteq D \setminus \{z_i\}$), then we have $\phi_{z_i} = 0$. From the expression in Theorem 8, we must have $\mathtt{R}_{i,m} = 0$ for all $0 \le m \le N$ and $\mathtt{G}_{i,\ell} = 0$ for all $0 \le \ell \le N-1$ (as these are non-negative quantities). Since $\widetilde{\mathtt{G}}_{i,\ell} \le \mathtt{G}_{i,\ell}$, we know that $\widetilde{\mathtt{G}}_{i,\ell} = 0$ for all $0 \le \ell \le N-1$. Hence, we have $\widehat{\phi}_{z_i}^{(M^\star)} = 0$.

Symmetry Axiom. Denote the condition $\mathtt{cond}_{\le m}$ as "within S, the $\min(\ell, K)$-th closest to the query example $x^{(\mathrm{val})}$ is among $\{x_k\}_{k=1}^m$. If two data points z_i, z_j are symmetry (i.e., $v(S \cup z_i) = v(S \cup z_j)$ for all $S \subseteq D \setminus \{z_i, z_j\}$), we must have $\widetilde{\mathtt{G}}_{i,\ell}^{(M^\star)} = \widetilde{\mathtt{G}}_{j,\ell}^{(M^\star)}$ since

$$\widetilde{\mathtt{G}}_{i,\ell}^{(M^\star)} = \sum_{S \subseteq D \setminus \{z_i\}, |S| = \ell, \mathtt{cond}_{\le M^\star}} [v(S \cup \{z_i\}) - v(S)]$$

Furthermore, we also have $\mathtt{R}_{i,m} = \mathtt{R}_{j,m}$ since

$$\mathtt{R}_{i,m} = \sum_{S \subseteq D \setminus \{z_i\}, |S| = K-1, \mathtt{cond}_{\le m-1}} [v(S \cup \{z_i\}) - v(S)]$$

which leads to $\widehat{\phi}_{z_i}^{(M^\star)} = \widehat{\phi}_{z_j}^{(M^\star)}$. □

G EVALUATION SETTINGS & ADDITIONAL EXPERIMENTS

We provide a summary of the content in this section for the convenience of the readers.

- Appendix G.1: General experiment settings (datasets and implementation details).
- Appendix G.2: Additional experiments for evaluating the influence of weights discretization.
- Appendix G.3: Experiment Settings and additional results for the runtime comparison of algorithms for computing/approximating WKNN-Shapley.
- Appendix G.4: Experiment Settings and additional results for noisy data detection task.
- Appendix G.5: Ablation study for the choice of $M^\star$ for the deterministic approximation algorithm from Section 4.2.
- Appendix G.6: Qualitative comparison between weighted and unweighted KNN-Shapley scores.

G.1 Experiment Settings

G.1.1 Datasets

An overview of the dataset information can be found in Table 2. Following the existing literature in data valuation (Ghorbani and Zou, 2019; Kwon and Zou, 2022; Jia et al., 2019b; Wang and Jia, 2023a; Wang et al., 2023), we preprocess datasets for ease of training. Following Kwon and Zou (2022), for Fraud, Creditcard, and all datasets from OpenML, we subsample the dataset to balance positive and negative labels. For these datasets, if they have multi-class, we binarize the label by considering $\mathbf{1}[y = 1]$. Following Wang et al. (2023), for the image dataset MNIST, CIFAR10, we apply a ResNet50 (He et al., 2016) that is pre-trained on the ImageNet dataset as the feature extractor. This feature extractor produces a 1024-dimensional vector for each image. For the sentence classification datasets AGNews and DBPedia, we use sentence embedding (Reimers and Gurevych, 2019) to extract features, resulting in 1024-dimensional vectors for each textual sample. We then standardize these extracted features using L2 normalization.

The size of each dataset we use is shown in Table 2. For some of the datasets, we use a subset of the full set. The validation data size we use is 10% of the training data size.

Dataset	Number of classes	Size of dataset	Source
Click	2	2000	`https://www.openml.org/d/1218`
Fraud	2	2000	Dal Pozzolo et al. (2015)
Creditcard	2	2000	Yeh and Lien (2009)
Apsfail	2	2000	`https://www.openml.org/d/41138`
Phoneme	2	2000	`https://www.openml.org/d/1489`
Wind	2	2000	`https://www.openml.org/d/847`
Pol	2	2000	`https://www.openml.org/d/722`
CPU	2	2000	`https://www.openml.org/d/761`
2DPlanes	2	2000	`https://www.openml.org/d/727`
MNIST	10	2000	LeCun (1998)
CIFAR10	10	2000	Krizhevsky et al. (2009)
AGnews	4	2000	Wang et al. (2021)
DBPedia	14	2000	Auer et al. (2007)

Table 2: A summary of datasets used in Section 5's experiments.

G.1.2 Implementation of Weighted KNN-Shapley

In Section 5 in the main text, the weights used in KNN are based on ℓ_2 distance between the training point and queried example, and then normalize all weights to $[0, 1]$. That is, the weight of a data point z_i is computed by

$$w_i := \frac{\left\|x_N - x^{(\mathrm{val})}\right\| - \left\|x_i - x^{(\mathrm{val})}\right\|}{\left\|x_N - x^{(\mathrm{val})}\right\| - \left\|x_1 - x^{(\mathrm{val})}\right\|}$$

The weights are then discretized by rounding to the nearest values that can be represented with b bits. That is, we create 2^b equally spaced points within the interval of $[0, 1]$, and round the weights to the closest point in the discretized space. We set the number of bits $b = 3$ in all experiments unless explicitly specified.

G.1.3 Details for Mislabel Data Detection Experiment

In the experiment of mislabeled data detection, we randomly choose 10% of the data points and flip their labels. Specifically, we flip 10% of the labels by picking an alternative label from the rest of the classes uniformly at random.

Remark 8 (**Baseline of Data Shapley**). *We note that several works have demonstrated that (unweighted) KNN-Shapley significantly outperforms the traditional Data Shapley (Pandl et al., 2021; Wang et al., 2023) in discerning data quality in tasks such as mislabeled data detection. Moreover, Data Shapley is highly inefficient as it requires ML models for many times, which is impractical for actual use. Hence, in this work, we omit the baseline of Data Shapley.*

G.2 Error From Weight Discretization

G.2.1 Value Deviation

We empirically study the difference between WKNN-Shapley computed based on the original continuous weights and the discretized weights. However, for continuous weights, it is computationally infeasible to compute the exact Data Shapley. Therefore, we instead look at the computed Shapley values' difference when using b bits and $b+1$ bits for $b = 1, 2, \ldots$. Figure 4 shows the results for ℓ_2 and ℓ_∞ error. That is, a point on the figure at x-axis b refers to the *error reduction* if using one more bit $b+1$. We have two observations here: **(1)** The error converges quickly as b increases and is near zero after $b \geq 5$. **(2)** The larger the dataset size N is, the smaller the error is. This interesting phenomenon is because the errors are dominated by the differences in the Shapley value computed for influential data points. When the dataset size is small, there are more influential data points since the performance of models trained on different data subsets can be significantly different from each other. On the other hand, when the dataset size is larger, there will be fewer influential points since most of the data subsets have a high utility (see Figure 5 for the visualization of the comparison between the distribution of data value scores).

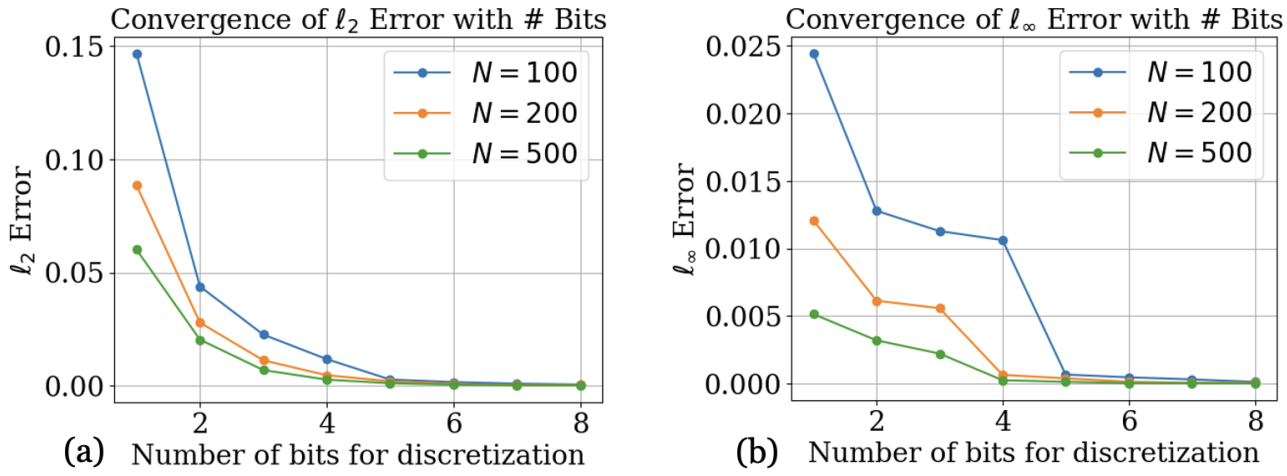

Figure 4: Convergence of the discretization error with the number of bits growth. The y-axis shows the ℓ_2 or ℓ_∞ norm of the difference between the Shapley values computed based on b bits and $b+1$ bits. The lower, the better. We use Fraud dataset from OpenML, and we use $K = 5$ here.

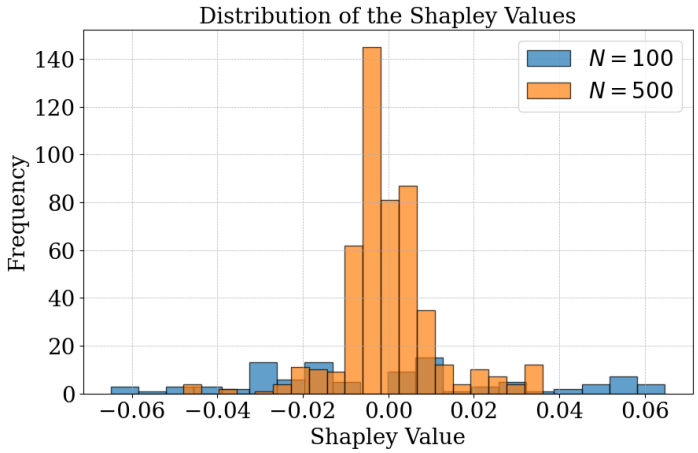

Figure 5: Distributions of WKNN-Shapley on different sizes of the subset of Fraud dataset from OpenML (the number of bits for discretization $b = 5$ and $K = 5$).

G.2.2 Performance on discerning data quality

In this section, we compare the performance between continuous and discretized weighted KNN-Shapley on the tasks of discerning data quality, specifically mislabeled data detection and noisy data detection. The experiment settings are the same as those described in Section 5.2 and Appendix G.4.

Comparison between continuous and discretized WKNN-Shapley. Due to the runtime complexity of $O(N^K)$ associated with the exact algorithm for computing WKNN-Shapley with continuous weights, it becomes unfeasible to compute WKNN-Shapley with continuous weights for values of $K > 2$. Thus, in this section, our focus remains on the comparative performance for $K = 2$. Table 3 shows the performance comparison on mislabeled data detection task, and Table 4 shows the performance comparison on noisy data detection task (with the same setting as stated in Appendix G.1). As we can see, WKNN-Shapley with discretized weights (third column) maintains a performance closely aligned with its counterpart that uses continuous weights (first column). This observation further validates that weight discretization only has a small influence on the efficacy of WKNN-Shapley in differentiating between high- and low-quality data points.

Comparison of discretized weighted KNN-Shapley with different discretization bits. Figure 6 and 7 show the performance comparison of WKNN-Shapley with different number of bits for discretization b, on the task of mislabeled data detection and noisy data detection, respectively. As we can see, the performance of both the exact and deterministic approximation of WKNN-Shapley is relatively stable, regardless of the number of discretization bits.

	Weighted KNN-Shapley (Continuous Weights)	**Weighted KNN-Shapley (Continuous Weights, Monte Carlo Approximation)**	**Weighted KNN-Shapley (Discretized Weights, this work)**	**Weighted KNN-Shapley (Discretized Weights, deterministic approximation, this work)**
2DPlanes	0.853	0.851	0.854	0.851
CPU	0.809	0.791	0.809	0.796
Phoneme	0.729	0.609	0.747	0.742
Fraud	0.511	0.501	0.507	0.513
Creditcard	0.723	0.716	0.724	0.698
Vehicle	0.733	0.636	0.72	0.769
Click	0.707	0.609	0.707	0.719
Wind	0.81	0.769	0.813	0.809
Pol	0.973	0.978	0.996	1
MNIST	0.732	0.742	0.733	0.72
CIFAR10	0.742	0.64	0.729	0.722
AGNews	0.942	0.932	0.944	0.914
DBPedia	0.969	0.969	0.988	0.988

Table 3: AUROC scores of different variants of weighted KNN-Shapley for mislabeled data detection on benchmark datasets at $K = 2$. The higher the AUROC score is, the better the method is. The Monte Carlo approximation (second column) is a stochastic algorithm. However, since it is already computationally expensive for just a single execution, we only run it once.

	Weighted KNN-Shapley (Continuous Weights)	Weighted KNN-Shapley (Continuous Weights, Monte Carlo Approximation)	Weighted KNN-Shapley (Discretized Weights, this work)	Weighted KNN-Shapley (Discretized Weights, deterministic approximation, this work)
2DPlanes	0.62	0.524	0.609	0.604
CPU	0.76	0.698	0.8	0.676
Phoneme	0.591	0.542	0.578	0.6
Fraud	0.831	0.722	0.831	0.622
Creditcard	0.493	0.507	0.533	0.489
Vehicle	0.569	0.529	0.533	0.44
Click	0.431	0.458	0.413	0.333
Wind	0.773	0.667	0.773	0.649
Pol	0.502	0.52	0.547	0.471
MNIST	0.68	0.671	0.676	0.678
CIFAR10	0.502	0.529	0.509	0.502
AGNews	0.508	0.508	0.472	0.444
DBPedia	0.447	0.451	0.443	0.514

Table 4: AUROC scores of different variants of weighted KNN-Shapley for noisy data detection on benchmark datasets at $K = 2$. The higher the AUROC score is, the better the method is. The Monte Carlo approximation (second column) is a stochastic algorithm. However, since it is already computationally expensive for just a single execution, we only run it once.

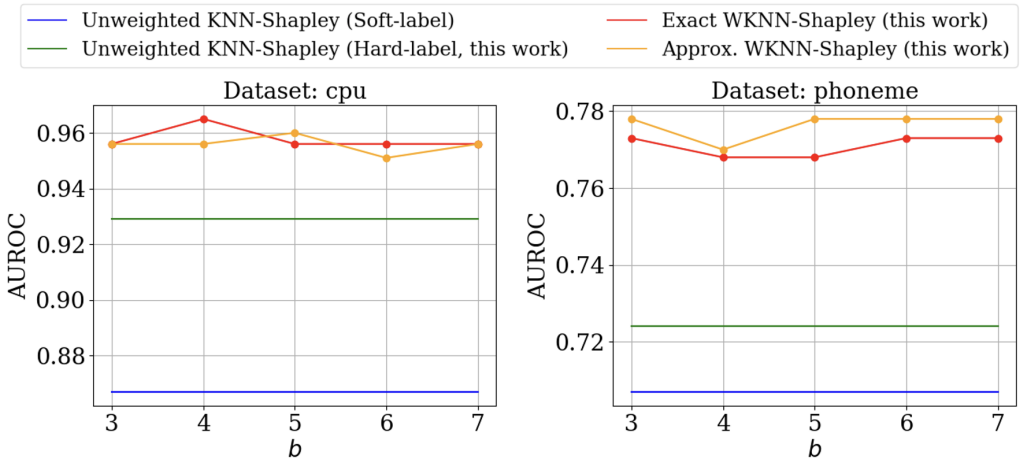

Figure 6: AUROC scores of different variants of KNN-Shapley for noisy data detection with different discretization bits b. The higher the curve is, the better the method is.

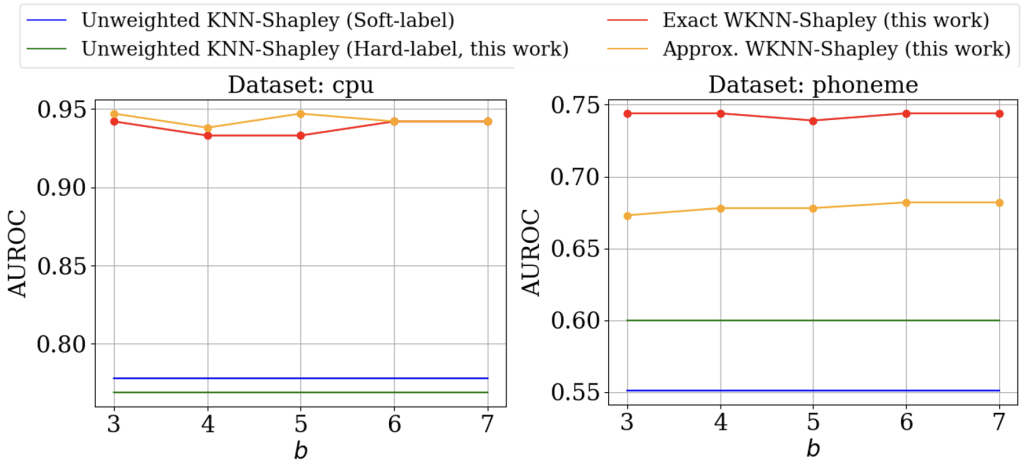

Figure 7: AUROC scores of different variants of KNN-Shapley for noisy data detection with different discretization bits b. The higher the curve is, the better the method is.

G.3 Settings & Additional Experiments for Runtime Comparison

Detailed Settings. For the runtime comparison experiment in Section 5.1, we follow similar experiment settings from prior study (Kwon and Zou, 2023) and use a synthetic binary classification dataset. To generate the synthetic dataset, we sample data points from a 2-dimensional standard Gaussian distribution, and the labels are assigned based on the sign of the sum of the two features. We note that the dataset dimension has minimal impact on the runtime of WKNN-Shapley compared with dataset size N, since the dataset dimension only affects the runtime of computing the distance between data points. All experiments were conducted on a 32-Core 2.6 GHz Intel Skylake CPU Processor.

We present additional experimental results comparing runtimes, further expanding on Section 5.1. We vary both K, the KNN hyperparameter, and b, the bit count for discretization.

Figure 8 shows the runtime comparison for our exact method and deterministic approximation for WKNN-Shapley, considering different number of bits b for discretization. As we can see, although the runtime increases with more bits for discretization, our algorithms, even at $b = 7$, demonstrate a $> 10^4$ times of improvement over both the exact computation and approximation algorithm introduced in Jia et al. (2019a).

Figure 9 shows the runtime comparison between our exact WKNN-Shapley computation algorithm and the $O(N^K)$ algorithm from Jia et al. (2019a), considering different choices of K. Since $K = 10$ for the baseline algorithm is computationally infeasible even for very small N (e.g., 20), we do not show the curve here. As we can see, our algorithm's curves for $K = 3$, $K = 5$, and $K = 10$ exhibit a relatively modest ascent in runtime, staying well below 10^6 seconds even at 100,000 training points. In contrast, the algorithm from Jia et al. (2019a) witnesses a steeper rise. This distinction is expected given that our algorithm's runtime scales at $O(K^2)$, whereas the one from Jia et al. (2019a) features an exponential time complexity with respect to K. Figure 10 shows the runtime comparison between our deterministic approximation algorithm and the Monte Carlo-based approximation algorithm (from Jia et al. (2019a)) for WKNN-Shapley. As we can see, our approximation algorithm is around $> 10^4$ times faster than the Monte Carlo algorithm for achieving the same error bound.

Remark 9. *We note that, as a training-free algorithm, KNN-Shapley exhibits a significant advantage in its computational efficiency compared with approaches such as Data Shapley/Banzhaf which requires many model retraining. For instance, as reported in Data Banzhaf's official Github repo*[5]*, it takes around 5 CPU hours to train 10,000 very small MLP models on different subsets of a tiny, size-200 dataset! On the contrary, it only takes a few seconds for KNN-Shapley under the same setting.*

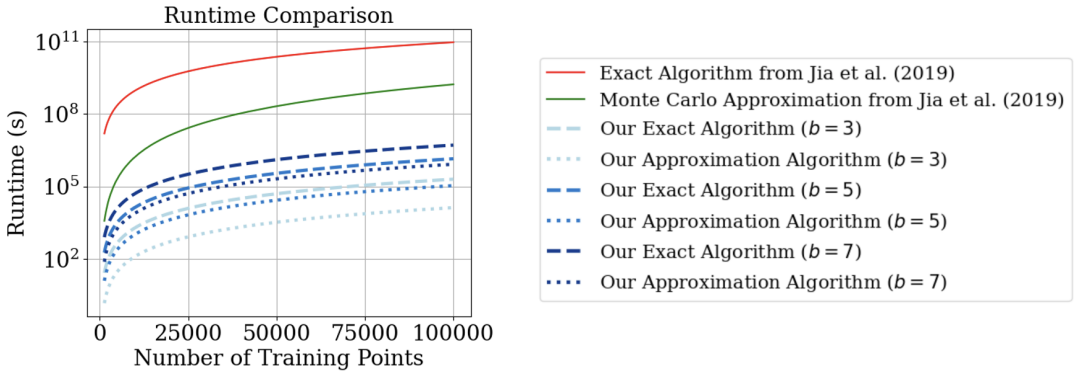

Figure 8: Runtime comparison between our exact and approximation algorithms for WKNN-Shapley in Section 4, and those from Jia et al. (2019a), across varying training data sizes N. We set $K = 5$ here for all methods. For our algorithms from Section 4, we vary the number of bits for discretization. All other settings are the same as Figure 2 in the maintext.

[5] https://github.com/Jiachen-T-Wang/data-banzhaf

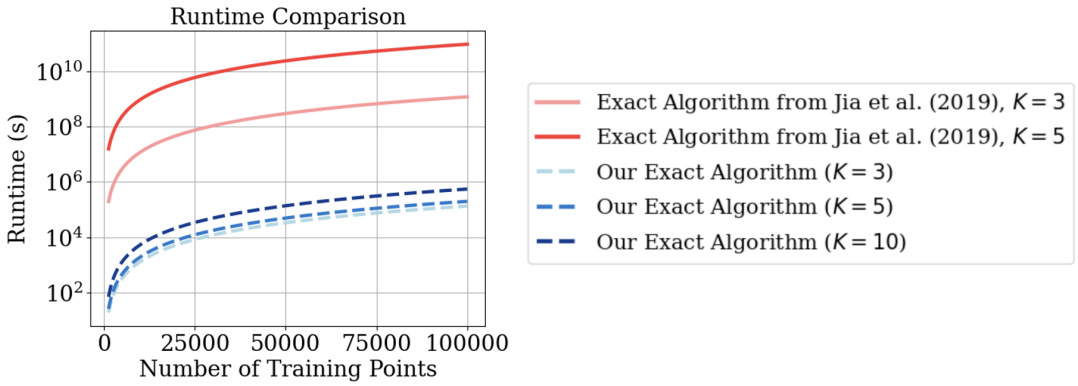

Figure 9: Runtime comparison between our exact WKNN-Shapley computation algorithm in Section 4.1, and the $O(N^K)$ from Jia et al. (2019a), across varying training data sizes N. We set $b = 3$ here for weights discretization in our algorithm. We vary and compare the runtime for different choices of K. All other settings are the same as Figure 2 in the maintext.

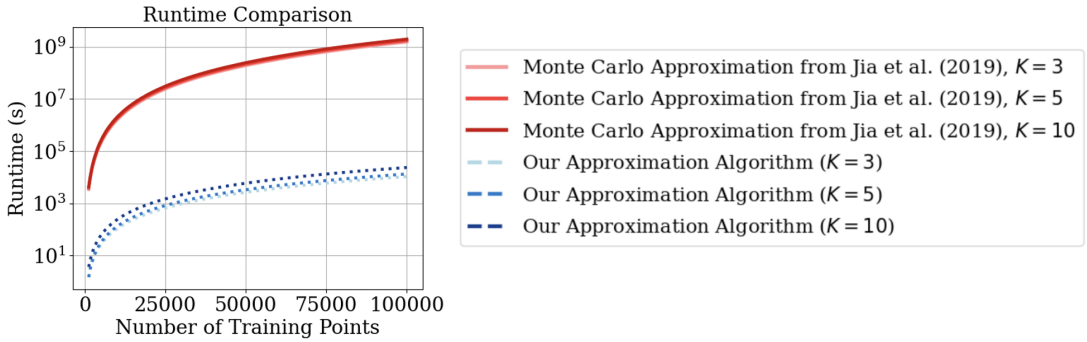

Figure 10: Runtime comparison between our deterministic WKNN-Shapley approximation algorithm in Section 4.2, and the Monte Carlo approximation from Jia et al. (2019a), across varying training data sizes N. We set $b = 3$ here for weights discretization in our algorithm. We vary and compare the runtime for different choices of K. All other settings are the same as Figure 2 in the maintext.

G.4 Application on Noisy Data Detection

Settings. In the experiment of noisy data detection, we randomly choose 10% of the data points and add strong noise to their features. Specifically, we add zero-mean Gaussian noise to data features, where the standard deviation of the Gaussian noise added to each feature dimension is equal to the average absolute value of the feature dimension across the full dataset. Similar to the task of mislabeled data detection, we use AUROC as the performance metric on noisy data detection tasks.

Table 5 shows the AUROC scores across the 13 benchmark datasets we experimented on when $K = 5$ and number of bits for discretization $b = 3$. Similar to the results for mislabeled data detection, we can see that both exact and approximated WKNN-Shapley significantly outperform the unweighted KNN-Shapley (either soft-label or hard-label) across most datasets, attributable to WKNN-Shapley's ability to more accurately differentiate between bad and good data based on the additional information of the proximity to the queried example. We can also see the similar encouraging result that the approximated WKNN-Shapley achieves performance comparable to, and sometimes even slightly better than, the exact WKNN-Shapley across the majority of datasets. This is likely attributable to its favored property in preserving the fairness properties of its exact counterpart.

In Figure 11, we show similar result on the task of noisy data detection that, compared to unweighted KNN-Shapley, WKNN-Shapley maintains notably stable performance across various choices of K, particularly for larger values of K. This is attributable to the additional weighting information incorporated in WKNN-Shapley.

	Soft-label KNN-Shapley (Jia et al. (2019))	**Hard-label KNN-Shapley (this work)**	**Exact WKNN-Shapley (this work)**	**Approximated WKNN-Shapley (this work)**
2DPlanes	0.556	0.498	**0.733**	0.68
CPU	0.778	0.769	0.942	**0.947**
Phoneme	0.551	0.6	**0.744**	0.673
Fraud	0.862	0.858	0.911	**0.916**
Creditcard	0.453	0.422	**0.653**	0.636
Vehicle	0.511	0.52	0.916	**0.933**
Click	0.44	0.444	**0.711**	0.662
Wind	0.782	0.804	0.849	**0.853**
Pol	0.493	0.502	**0.836**	0.804
MNIST	0.782	0.538	**0.911**	**0.911**
CIFAR10	0.533	0.418	0.8	**0.822**
AGNews	0.481	0.531	**0.559**	0.543
DBPedia	0.482	0.498	**0.58**	0.576

Table 5: AUROC scores of different variants of KNN-Shapley for noisy data detection tasks on various datasets. The higher the AUROC score is, the better the method is.

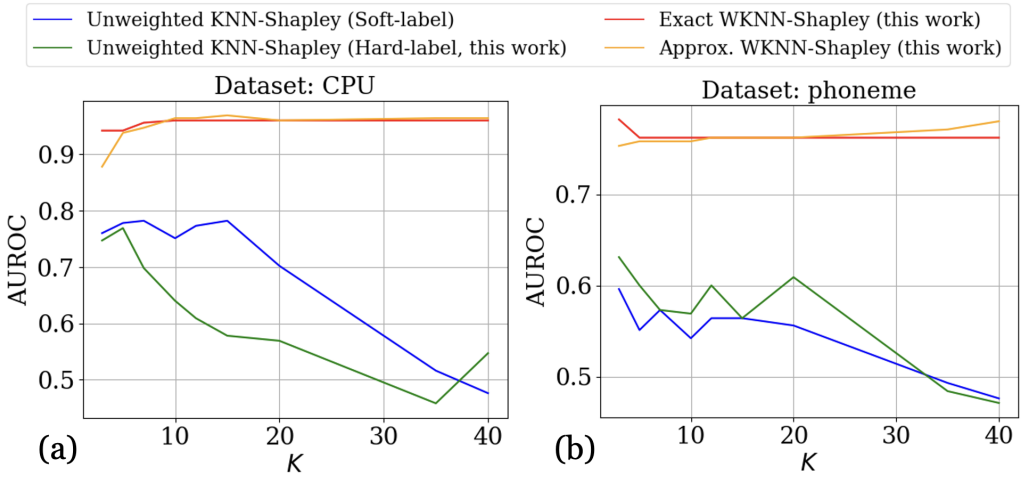

Figure 11: AUROC scores of different variants of KNN-Shapley for noisy data detection with different Ks. The higher the curve is, the better the method is.

G.5 Ablation Study for the choice of $M^\star$ for the deterministic approximation algorithm

In this section, we evaluate the performance variation when we pick different $M^\star$ for our deterministic approximation algorithm in Section 4.2. Specifically, we choose different values of $M^\star$ and plot the performance variation in mislabeled/noisy data detection task with different error bound $\varepsilon(M^\star)$. Note that $\varepsilon(M^\star) = 0$ corresponds to the exact WKNN-Shapley. We also highlight the location of $\varepsilon(\sqrt{N})$, i.e., the error bound for the $M^\star$ we set in the experiment. As we can see from Figure 12, the performance of the approximated WKNN-Shapley is highly stable across a wide range of choices of $M^\star$s. Hence, we set $M^\star = \sqrt{N}$ in all of the experiments instead of following the adaptive procedure of selecting $M^\star$ mentioned in Appendix D.4.

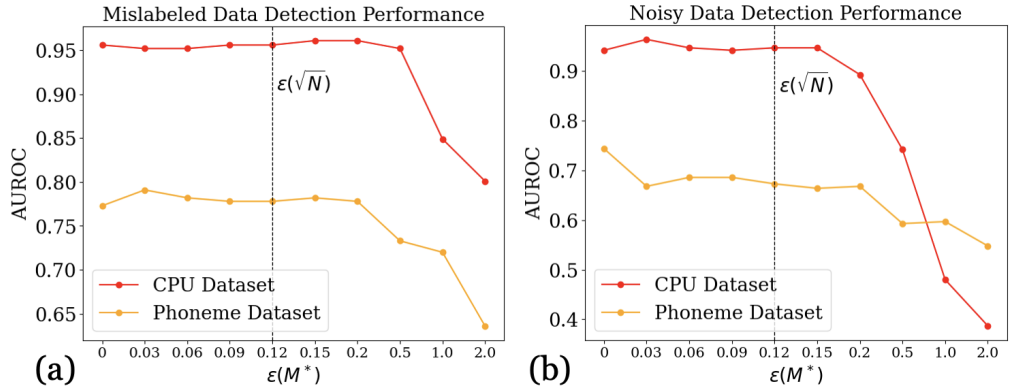

Figure 12: Performance variation of approximated WKNN-Shapley on (a) mislabeled data detection and (b) noisy data detection across different choice of $M^\star$. For a more direct comparison, we plot x-axis as the theoretical error bound $\varepsilon(M^\star)$ derived in Theorem 12.

G.6 Qualitative Comparison between Weighted and Unweighted KNN-Shapley

In our experiments, we show that weighted KNN-Shapley significantly outperforms unweighted KNN-Shapley in discerning data quality. This can likely be attributed to WKNN-Shapley's adept ability to more accurately differentiate between bad and good data based on the proximity to the queried example. In this section, we present a more detailed qualitative analysis highlighting why WKNN-Shapley outperforms unweighted KNN-Shapley in discerning data quality.

Figure 13 shows the value score distribution of unweighted and weighted KNN-Shapley of 50 data points, where 5 of them are being mislabeled. The KNN-Shapley scores are computed with respect to a single validation point. As we can see, compared with the unweighted KNN-Shapley, WKNN-Shapley exhibits a much higher variation in value scores of different data points. More importantly, WKNN-Shapley can better differentiate the quality between the data points that are near the validation point. As we can see from the figure, the two mislabeled points that are the closest to the validation point (index **49** and **50**) receive much lower WKNN-Shapley scores compared with those benign data points that have a different label as the validation point (index **41-48**). On the other hand, unweighted KNN-Shapley assigns almost the same negative values for all data points that are close to the validation point but has a different label (index **41-50**), regardless of whether they are benign or mislabeled.

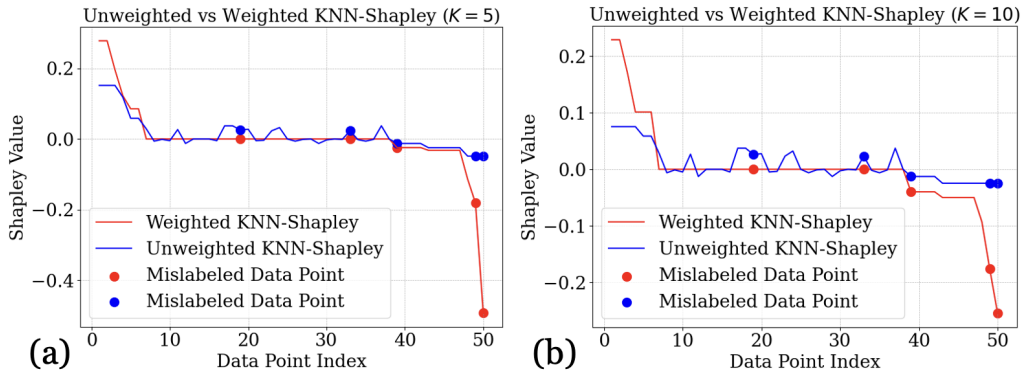

Figure 13: Distribution of unweighted and weighted KNN-Shapley scores at (a) $K = 5$ and (b) $K = 10$ for 50 data points from CPU dataset, where 5 of them are mislabeled data points.