New Algorithms for Monotone Classification

Yufei Tao
Chinese University of Hong Kong
Hong Kong, China
taoyf@cse.cuhk.edu.hk

|Yu Wang|*
Chinese University of Hong Kong
Hong Kong, China

ABSTRACT

In *monotone classification*, the input is a set P of n points in d-dimensional space, where each point carries a label 0 or 1. A point p *dominates* another point q if the coordinate of p is at least that of q on every dimension. A *monotone classifier* is a function h mapping each d-dimensional point to $\{0, 1\}$, subject to the condition that $h(p) \geq h(q)$ holds whenever p dominates q. The classifier h *mis-classifies* a point $p \in P$ if $h(p)$ is different from the label of p. The *error* of h is the number of points in P mis-classified by h. The objective is to find a monotone classifier with a small error. The problem is fundamental to numerous database applications in entity matching, record linkage, and duplicate detection.

This paper studies two variants of the problem. In the first *active* version, all the labels are hidden in the beginning; an algorithm must pay a unit cost to *probe* (i.e., reveal) the label of a point in P. We prove that $\Omega(n)$ probes are necessary to find an optimal classifier even in one-dimensional space ($d = 1$). On the other hand, given an arbitrary $\epsilon > 0$, we show how to obtain (with high probability) a monotone classifier whose error is worse than the optimum by at most a $1 + \epsilon$ factor, while probing $\tilde{O}(w/\epsilon^2)$ labels, where w is the dominance width of P and $\tilde{O}(.)$ hides a polylogarithmic factor. For constant ϵ, the probing cost matches an existing lower bound up to an $\tilde{O}(1)$ factor. In the second *passive* version, the point labels in P are explicitly given; the goal is to minimize CPU computation in finding an optimal classifier. We show that the problem can be settled in time polynomial to both d and n.

CCS CONCEPTS

• **Theory of computation → Approximation algorithms analysis**; **Active learning**.

KEYWORDS

Active Learning; Monotone Classification; Entity Matching

*Yu Wang, who passed away in 2019, was a Ph.D. student of Yufei Tao at the Chinese University of Hong Kong. The paper is in memory of Yu's contributions in the early phase of this work.

PODS '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458324

ACM Reference Format:
Yufei Tao and |Yu Wang|. 2021. New Algorithms for Monotone Classification. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3452021.3458324

1 INTRODUCTION

We consider two problems fundamental in entity matching, record linkage, and duplicate detection. A common goal behind these applications is to decide whether a pair of objects is a *match* or a *non-match* based on their similarity scores on selected dimensions. This requires learning a function h that, given a set of similarity scores, outputs a binary verdict: 1 (i.e., accepting the corresponding objects as a match) and 0 (rejecting them). The function is *explainable* if it never accepts a less similar pair while rejecting a more similar pair. Finding such a function with high accuracy introduces various challenges. Next, we formalize two problems to tackle some of those challenges.

1.1 Problem Definitions

Active monotone classification. The input is a set P of n points in $\mathbb{R}^d$ where $d \geq 1$ is the dimensionality. Each point $p \in P$ is associated with a binary label, represented as $label(p)$ (= 0 or 1). Denote by $p[i]$ the coordinate of p on dimension $i \in [1, d]$.

A point p *dominates* another (different) point q if $p[i] \geq q[i]$ for all $i \in [1, d]$; note that $p \neq q$ implies that $p[i] > q[i]$ holds on at least one i. We will use $p \succcurlyeq q$ to indicate p dominating q.

A *classifier* is a function $h : \mathbb{R}^d \to \{0, 1\}$, namely, $h(p)$ maps a point $p \in \mathbb{R}^d$ to either label 0 or 1. Given a point $p \in P$, we say that h *correctly classifies* p if $h(p) = label(p)$, or *mis-classifiers* p otherwise. We define the *error* of h on P as

$$err_P(h) = |\{p \in P \mid h(p) \neq label(p)\}| \quad (1)$$

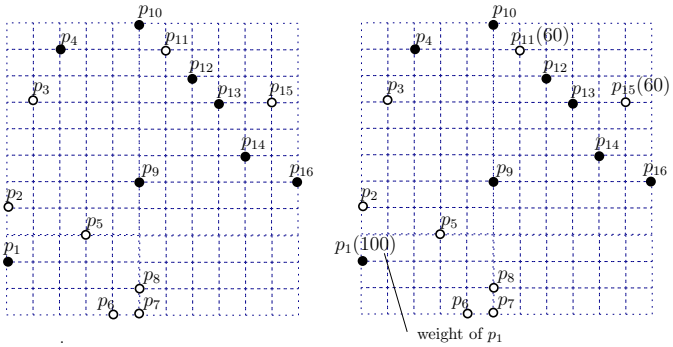
that is, the number of points in P mis-classified.

A classifier h is *monotone* if $h(p) \geq h(q)$ holds for any distinct points $p, q \in \mathbb{R}^d$ such that $p \succcurlyeq q$. Denote by $\mathbb{H}_{mono}$ the set of all possible monotone classifiers. Our objective is to find a monotone classifier with the minimum error on P. Note that the minimum error may *not* be zero because P does not need to obey monotonicity.

Define

$$k^* = \min_{h \in \mathbb{H}_{mono}} err_P(h) \quad (2)$$

namely, the minimum error of all monotone classifiers on P. A classifier $h \in \mathbb{H}_{mono}$ is *c-approximate* if $err_P(h) \leq ck^*$. A 1-approximate classifier is *optimal*.



(a) Problem 1 (figure from [25]) (b) Problem 2

Figure 1: Example input sets

As an example, consider Figure 1(a), which shows a 2D input set P where a black (or white) point has label 1 (or 0, respectively). Consider the monotone classifier h that maps (i) all the black points to 1 except p_1, and (ii) all the white points to 0 except p_{11} and p_{15}. Therefore, $err_P(h) = 3$. No other monotone classifiers have a smaller error on P; in other words, $k^* = 3$.

In the beginning, the labels of all the points in P are hidden. An algorithm can *probe* an arbitrary point $p \in P$ by asking an oracle to reveal $label(p)$. In the end, the algorithm must return a monotone classifier. The algorithm's *probing cost* is the total number of points probed. Obviously, with probing cost n, an algorithm acquires all the labels and, thus, can always return an optimal classifier (after sufficient CPU computation). The challenge, however, is to achieve the purpose with the minimum probing.

This brings us to the first problem studied in this work:

Problem 1 (active monotone classification): Given P and a value $\epsilon \geq 0$, find a $(1+\epsilon)$-approximate monotone classifier while minimizing the probing cost.

Although CPU efficiency is not an explicit concern in Problem 1, an algorithm suitable for practical use should have a tractable running time, i.e., polynomial to n, d, and $1/\epsilon$.

Passive weighted monotone classification. A *fully-labeled weighted set* refers to a set P of points in $\mathbb{R}^d$ where each point $p \in P$ is associated with

- (as before) $label(p)$, which is either 0 or 1;
- a positive finite real-valued weight, denoted as $weight(p)$.

Given a classifier $h \in \mathbb{H}_{mono}$, define its *weighted error* on P as:

$$w\text{-}err_P(h) = \sum_{p \in P} weight(p) \cdot \mathbb{1}_{h(p) \neq label(p)} \quad (3)$$

where, in general, $\mathbb{1}_\pi$ equals 1 if predicate π holds or 0 otherwise. Note that $w\text{-}err(h)$ captures the $err(h)$ in (1) as a special case where every p has weight 1.

Let us use Figure 1(b) to illustrate the above concepts. The input set P is the same as in Figure 1(a), but p_1 now carries a weight of 100, p_{11} and p_{15} have weight 60, and all the other points have weight 1. Consider the aforementioned monotone classifier h optimal under Problem 1. Recall that h correctly classifies all the points except p_1, p_{11}, and p_{15}. The weighted error of h on P is $w\text{-}err_P(h) = 100 + 60 + 60 = 220$. This is far from being optimal under Problem 2. Consider instead another monotone classifier h' that maps p_{10}, p_{12}, p_{16} to 1 and everything else to 0. It is easy to verify that $w\text{-}err_P(h) = 104$ (the weight sum of p_1, p_4, p_9, p_{13}, and p_{14}). This is actually an optimal classifier under Problem 2.

We now introduce the second problem studied in this work:

Problem 2 (passive weighted monotone classification): Given a fully-labeled weighted set P, find a monotone classifier with the smallest weighted error on P while minimizing the CPU time.

As before, set $n = |P|$. It is worth emphasizing that (i) while Problem 1 concerns only the probing cost, Problem 2 focuses exclusively on CPU time; (ii) no approximation is allowed in Problem 2.

Connections to similarity-based matching. Consider two (possibly infinite) sets of objects: X and Y. We want to devise a method that, given $x \in X$ and $y \in Y$, can determine whether x and y form a match. This is a core problem behind many applications, e.g., matching the advertisements on Amazon with those on eBay selling the same product, identifying records from two databases describing the same entity, determining whether a document plagiarizes another, etc. A popular approach is to calculate the similarity scores of x and y on a number d of carefully-chosen metrics: $sim_1(x, y), sim_2(x, y), ..., sim_d(x, y)$ (a higher score indicates greater similarity). We want to learn a function that casts a verdict (on whether x matches y) based on these scores. If we define $p_{x,y} = (sim_1(x, y), ..., sim_d(x, y))$, such a function can be regarded as a classifier that maps $\mathbb{R}^d$ to $\{0, 1\}$ with 1 representing a match.

To carry out learning, we are given a sample set $S \subseteq X \times Y$ following some unknown distribution $\mathcal{D}$ over $X \times Y$. Define

$$P = \{p_{x,y} \mid (x, y) \in S\}$$

where each point $p_{x,y}$ carries a label 1 if x and y are a match, and 0 otherwise. The goal of learning is to find a classifier $h : \mathbb{R}^d \to \{0, 1\}$ to minimize $err_P(h)$. Such a classifier is expected to perform well on a general object pair (x, y) (not necessarily in S) drawn from $\mathcal{D}$.

Problems 1 and 2 suit different scenarios under the above settings. Problem 1 applies when the label of $p_{x,y}$ is costly to obtain. For example, determining whether two advertisements are about the same product requires human inspection, which is both time and money consuming; it can be prohibitively expensive to do so for all advertisement pairs in S. Thus, the chief objective is to find an accurate classifier by minimizing the amount of label probing. Problem 2, on the other hand, applies when all the labels are already available (e.g., this happens usually when S is small). The goal is then to find an optimal classifier by minimizing computation.

Demanding the monotonicty of h avoids the strange situation of classifying (x, y) as a non-match but (x', y') as a match when $p_{(x,y)} \succcurlyeq p_{(x',y')}$. This oddity is difficult to explain because the former pair is at least as similar as the latter on every dimension.

Math conventions. Given two non-negative integers x and y, we use notation $x \leq_\epsilon y$ to represent the fact that $x \leq (1+\epsilon)y$ where $\epsilon \geq 0$. Given a real value $x > 0$, we use $\log x$ as a short form for $1 + \log_2 x$.

1.2 Previous Results

Tao [25] showed that a key parameter characterizing the hardness of Problem 1 is the *dominance width* w of P. Formally, w is the size of the largest subset $S \subseteq P$ such that no two points in S have a dominance relationship (e.g., for the input set P in Figure 1(a), $w = 6$, as is witnessed by $S = \{p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{16}\}$). The focus of [25] was to study algorithms with small "expected errors". Formally, let A be a randomized algorithm and h be the classifier it returns on P. As h is a random variable, so is $err_P(h)$, whose expectation is the *expected error* of A on P. The results of [25] are:

- There is an algorithm that probes $O(w \log \frac{n}{w})$ labels in expectation, and has an expected error of at most $2k^*$, where k^* is the optimal error (see (2)).
- Any algorithm with expected error ck^*, regardless of the constant $c \geq 1$, must probe $\Omega(w \log \frac{n}{(1+k^*)w})$ labels in expectation. This matches the above upper bound up to a small factor (and essentially tight when k^* is small).

However, the error guarantee of Tao's algorithm [25] is weak in two ways. First, it fails to come close to $(1+\epsilon)k^*$ for an arbitrarily small $\epsilon > 1$ (the approximation ratio 2 is already tight). Second, even the $2k^*$ error bound holds only in expectation, as opposed to with high probability.

Currently, the A^2 algorithm (see [2, 4, 9, 15] for the history of its development) is the best approach for finding a monotone classifier of error at most $(1+\epsilon)k^*$ with high probability. The algorithm probes $\tilde{O}(\theta \cdot \lambda \cdot \frac{1}{\epsilon^2})$ labels, where λ is the *VC dimension* [22] of P under the set $\mathbb{H}_{mono}$ of monotone classifiers, θ is the *disagreement coefficient* [15] of $\mathbb{H}_{mono}$ on P, and the notation $\tilde{O}(.)$ hides a factor polylogarithmic to d, n, and $1/\epsilon$. We do not need the exact meanings of λ and θ, except to note that both of λ and θ are $\Omega(w)$, as shown in [25]. This means that the probing bound of the A^2 algorithm is $\Omega(w^2/\epsilon^2)$ at the very best.

No previous work exists on Problem 2. A naive solution is to examine every possible subset $S \subseteq P$. Specifically, we check whether mapping the entire S to 1 and the entire $P \setminus S$ to 0 makes a monotone classifier h. If so, we proceed to compute $w\text{-}err_P(h)$ and maintain the best classifier seen so far. This shows that Problem 2 is decidable, but the solution's running time is exponential to n. Later, we will prove that Problem 2 can be settled in polynomial time.

The work of [25] also considered a variant of Problem 1 called *active monotone classification with exceptions*, which differs from Problem 1 in how the expected error of an algorithm A is calculated. Finally, it is worth mentioning that entity matching, record linkage, duplicate detection, and other similar problems have attracted a huge amount of attention from the system community; see [1, 3, 5–7, 11, 12, 17–21, 24, 26] and the references therein. When viewed through the lens of theory, however, those papers do not provide formal findings that can rival the aforementioned results.

1.3 Our Results

Our first contribution is to a new hardness result on active monotone classification:

Theorem 1. *For Problem 1, any algorithm promising to find an optimal classifier with probability over 2/3 must probe $\Omega(n)$ labels in expectation. This is true even if the dimensionality is 1.*

As mentioned, with n probes, an algorithm can reveal the labels of all the points in P, after which it can always discover an optimal monotone classifier (CPU time is not a concern here). Theorem 1 suggests that the naive approach is already optimal up to a constant factor. This justifies the studies on finding approximate classifiers with fewer probes. Our second contribution is a new algorithm that achieves an approximation ratio arbitrarily close to 1:

Theorem 2. *For any $\epsilon \in (0, 1]$, with probability at least $1 - 1/n^2$ we can solve Problem 1 by probing $O(\frac{w}{\epsilon^2} \cdot \log \frac{n}{w} \cdot \log n)$ points, where n, d, and w are the size, dimensionality, and dominance width of the input set P, respectively.*

Several remarks are in order:

- The failure probability $1/n^2$ can be reduced to $1/n^c$ for an arbitrarily large constant $c > 2$ without affecting the theorem's asymptotic claim.
- If the optimal error k^* (see (2)) is 0, our algorithm finds an optimal classifier with high probability.
- When $k^* > 0$, the ability of returning a classifier with error at most $(1+\epsilon)k^*$ with high probability implies that our algorithm can ensure an expected error $(1+\epsilon)k^*$, noticing that n is the maximum possible error for a classifier[1].
- The probing cost bound holds for every run of our algorithm (with probability 1). For a constant ϵ, the bound becomes $O(w \log \frac{n}{w} \cdot \log n)$, which is higher than the lower bound $\Omega(w \log \frac{n}{(1+k^*)w})$ of [25] (see Section 1.2) by only a small factor.
- Compared to the existing results (Section 1.2), our algorithm enjoys much stronger error guarantees than [25] and improves the probing cost of the A^2-algorithm by nearly a factor of $O(w)$.

The algorithm of Theorem 2, if implemented directly, may entail expensive CPU overhead (exponential to w). To improve CPU efficiency, we observe a connection between active and passive monotone classification:

Theorem 3. *If Problem 2 can be settled in $T_{prob2}(d, n)$ time, we can solve Problem 1 with the guarantees in Theorem 2 and the CPU time $\tilde{O}(dn^2 + n^{2.5} + w/\epsilon^2) + T_{prob2}(d, N)$, where $N = O(\frac{w}{\epsilon^2} \cdot \log n \cdot \log \frac{n}{w})$ and $\tilde{O}(.)$ hides a factor polylogarithmic to d, n, and $1/\epsilon$.*

In other words, as long as passive weighted monotone classification can be settled in polynomial time, the proposed algorithm in Theorem 2 runs in time polynomial to d, n, and $1/\epsilon$. Our final contribution is to settle Problem 2 in polynomial time by connecting it to the max-flow problem (to be reviewed in Section 2):

[1] Suppose that the goal is to achieve expected error $1.5k^*$. We can set ϵ to 0.4, in which case the expected error is at most $1.4k^*(1 - 1/n^2) + n/n^2 < 1.5k^*$, assuming $n \geq 10$.

Theorem 4. *Problem 2 can be solved in $O(dn^2) + T_{maxflow}(n)$ time, where n and d are the size and dimensionality of the input set P, respectively, and $T_{maxflow}(n)$ is the time of solving the max-flow problem on a graph of n vertices.*

The results of this paper complement those of [25] in providing a systematic theory on passive and active monotone classification.

2 PRELIMINARIES

Estimation up to an absolute error. Let us start with a fundamental lemma (proof in the appendix):

Lemma 5. *Let $X_1, X_2, ..., X_t$ be t independent Bernoulli variables such that $\mathbf{Pr}[X_i = 1] = \mu$ for each $i \in [1, t]$ (and hence, $\mathbf{Pr}[X_i = 0] = 1 - \mu$). For any $\phi \in (0, 1]$ and $\delta \in (0, 1]$,*

$$\mathbf{Pr}\left[\left|\mu - \frac{1}{t}\sum_{i=1}^{t} X_i\right| \geq \phi\right] \quad \leq \quad \delta \tag{4}$$

as long as $t \geq \lceil \max\{\frac{\mu}{\phi^2}, \frac{1}{\phi}\} \cdot 3 \ln \frac{2}{\delta} \rceil$.

Consider the input set P of Problem 1. Suppose that we want to estimate the number n_π of points in P satisfying an (arbitrary) predicate π. We can draw, with replacement, a set S of $t = O(\frac{1}{\phi^2} \log \frac{1}{\delta})$ points from P uniformly at random. If x is the number of points in S satisfying π, Lemma 5 assures us that $(x/t) \cdot n$ approximates n_π up to absolute error ϕn, with probability at least $1 - \delta$.

As a further corollary, given any $h \in \mathbb{H}_{mono}$, we can utilize S to estimate $err_P(h)$ (see (1)) up to absolute error ϕn, by formulating π as follows: a point $p \in P$ satisfies π if and only if $label(p) \neq h(p)$.

Chain decomposition and dominance width. Let P be a set of points in $\mathbb{R}^d$. A subset $S \subseteq P$ is an *anti-chain* if there do not exist any distinct points $p, q \in S$ such that $p \succcurlyeq q$. The dominance width w of P, introduced earlier in Section 1.2, is the essentially size of the largest anti-chain.

In contrast, a *chain* is a subset $C \subseteq P$ such that it is possible to arrange the points in C into a sequence $p_1, p_2, ..., p_{|C|}$ such that $p_{i+1} \succcurlyeq p_i$ for each $i \in [2, |C|]$. A *chain decomposition* of P divides P into t disjoint chains $C_1, C_2, ..., C_t$, namely, $\bigcup_{i=1}^{t} C_i = P$ and $C_i \cap C_j = \emptyset$ for any distinct $i, j \in [1, t]$.

Dilworth's theorem [10] gives a connection between the dominance width and chain decompositions: *w is the smallest number of chains that P can be partitioned into.* Specifically, there must exist a chain decomposition with w chains; conversely, every chain decomposition must have at least w chains.

To illustrate these concepts, let us look at the input set P in Figure 1(a) again. We can divide P into 6 chains: $C_1 = \{p_1, p_2, p_3, p_4, p_{10}\}$, $C_2 = \{p_{11}\}$, $C_3 = \{p_5, p_9, p_{12}\}$, $C_4 = \{p_{16}\}$, $C_5 = \{p_{13}\}$, and $C_6 = \{p_6, p_7, p_8, p_{14}, p_{15}\}$. No chain decompositions can have fewer chains, as is evidenced by the size 6 of the anti-chain $\{p_{10}, p_{11}, p_{12}, p_{16}, p_{13}, p_{14}\}$. The dominance width w of P is 6.

We prove in the appendix:

Lemma 6. *Given a set P of n points in $\mathbb{R}^d$, we can compute in $O(dn^2 + n^{2.5})$ time a chain decomposition of P containing w chains.*

The max-flow problem. The *max-flow problem* is a classic problem in computer science. The input is a directed graph $G = (V, E)$ where V contains a *source* vertex $v_\sqsubset$ with in-degree 0 and a *sink* vertex $v_\sqsupset$ with out-degree 0. Every edge $e \in E$ is associated with a non-negative value $cap(e)$ called its *capacity*. Sometimes, we may represent an edge from vertex u to vertex v as (u, v); accordingly, $cap(u, v)$ denotes the edge's capacity.

A *flow* on G is a function $F : E \to [0, \infty)$ satisfying:

- (capacity constraint) $F(e) \leq cap(e)$ for every $e \in E$;
- (conservation constraint) for every vertex $u \in V$ other than $v_\sqsubset$ and $v_\sqsupset$:

$$\sum_{\text{in-edge } e \text{ of } u} F(e) \quad = \quad \sum_{\text{out-edge } e \text{ of } u} F(e).$$

The *value* of the flow F equals

$$\sum_{\text{out-edge } e \text{ of } v_\sqsubset} F(e).$$

The goal of the problem is to find an F of the maximum value.

Numerous algorithms exist for solving the problem with various time complexities (see [13, 14] for an extensive list). In particular, Goldberg and Tarjan [14] gave an algorithm of $O(|V|^3)$ time, applying which the function $T_{maxflow}(n)$ in Theorem 4 evaluates to $O(n^3)$.

A *source-sink cut* is a pair $(V_\sqsubset, V_\sqsupset)$ where

- $V_\sqsubset$ and $V_\sqsupset$ are disjoint subsets of V such that $V_\sqsubset \cup V_\sqsupset = V$;
- $v_\sqsubset \in V_\sqsubset$ and $v_\sqsupset \in V_\sqsupset$.

The *capacity* of the source-sink cut is the capacity sum of all the "cross edges" $(u, v) \in E$ satisfying $u \in V_\sqsubset$ and $v \in V_\sqsupset$, or formally:

$$\sum_{(u,v) \in E \cap (V_\sqsubset \times V_\sqsupset)} cap(u, v).$$

The following result is known as the *max-flow min-cut theorem*:

Lemma 7 (Theorem 26.7 of [8]). *The maximum value of all flows equals the minimum capacity of all source-sink cuts.*

For our purposes, it will be easier to work with an alternative interpretation of minimum source-sink cuts. We call a subset $E_{cut} \subseteq E$ a *cut-edge set* if the removal of E_{cut} disconnects s and t, i.e., no paths exist from s to t in the graph induced by $E \setminus E_{cut}$. Define the *weight* of E_{cut} as the capacity sum of all the edges in E_{cut}, or formally:

$$weight(E_{cut}) \quad = \quad \sum_{e \in E_{cut}} cap(e). \tag{5}$$

The following should be folklore (see the appendix for a proof):

Lemma 8. *The minimum capacity of all source-sink cuts equals the minimum weight of all cut-edge sets.*

Lemmas 7 and 8 together indicate that the maximum value of flows is precisely the minimum weight of cut-edge sets. Any algorithm solving the max-flow problem can be used to construct an optimal cut-edge set with the smallest weight with $\tilde{O}(|E|)$ extra time (the proof of Lemma 8 gives such a construction explicitly).

3 ACTIVE CLASSIFICATION IN 1D SPACE

In this section, we will discuss Problem 1 (active monotone classification) for $d = 1$ and a strictly positive ϵ. Each point $p \in P$ is simply a real value. In 1D space, a monotone classifier h has a very specific form:

$$h(p) \quad = \quad \begin{cases} 1 & \text{if } p > \tau \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Note that each $h \in \mathbb{H}_{mono}$ is parameterized by a real value τ; our discussion will make the parameter explicit by representing the classifier as h^τ. Even though $\mathbb{H}_{mono}$ has an infinite size, it suffices to consider a set of "effective" classifiers:

$$\mathbb{H}_{mono}(P) \quad = \quad \{h^\tau \mid \tau \in P \text{ or } \tau = -\infty\}. \tag{7}$$

Every other classifier has the same error as one of the classifiers in $\mathbb{H}_{mono}(P)$. We will prove:

Lemma 9. *For any $\epsilon \in (0, 1]$ and any $\delta > 0$, with probability at least $1 - \delta$ we can solve Problem 1 in 1D space by probing $O((1/\epsilon^2) \cdot \log n \cdot \log \frac{n}{\delta})$ points and paying CPU time $\tilde{O}(1/\epsilon^2)$, where $\tilde{O}(.)$ hides a factor polylogarithmic to n, $1/\epsilon$, and $1/\delta$.*

3.1 The Rationale

Our algorithm will produce (with probability at least $1-\delta$) a function $f : \mathbb{H}_{mono} \to [0, \infty)$ that ensures:

> **the ϵ-comparison property:** for any classifiers $h^x, h^y \in \mathbb{H}_{mono}$, $f(h^x) \leq f(h^y)$ implies $err_P(h^x) \leq_\epsilon err_P(h^y)$.

See Section 1.1 for the meaning of $\leq_\epsilon$. Once f is known, we can simply return the classifier $h^\tau \in \mathbb{H}_{mono}(P)$ with the smallest $f(h^\tau)$, which is guaranteed to have an error on P at most $k^*(1 + \epsilon)$, where k^* is the optimal error (see (2)).

Specifically, for any $h^\tau \in \mathbb{H}_{mono}$, our function f guarantees

$$|f(h^\tau) - err_P(h^\tau)| \leq \epsilon |P|/64 \tag{8}$$

and

$$err_P(h^\tau) \cdot \left(1 - \frac{\epsilon}{4}\right) + \Delta \leq f(h^\tau) \leq err_P(h^\tau) \cdot \left(1 + \frac{\epsilon}{4}\right) + \Delta \tag{9}$$

where Δ satisfies

$$|\Delta| \quad \leq \quad \epsilon |P|/16. \tag{10}$$

If the exact value of Δ *were* available, $f(h^\tau) - \Delta$ *would* serve as an estimate of $err_P(h^\tau)$ with relative error at most $\epsilon/4$. However, as shown in [25] it is impossible to obtain such an estimate without a probing cost of $\Omega(|P|)$. Our key idea is to ensure (9) *without* knowing Δ precisely.

The ϵ-comparison property is a corollary of (9). Indeed, by applying (9), we get

$$\begin{aligned} err_P(h^x) \quad &\leq \quad \frac{1}{1 - \epsilon/4} \cdot (f(h^x) - \Delta) \\ (\text{by } f(h^x) \leq f(h^y)) \quad &\leq \quad \frac{1}{1 - \epsilon/4} \cdot (f(h^y) - \Delta) \\ (\text{applying (9) again}) \quad &\leq \quad \frac{1 + \epsilon/4}{1 - \epsilon/4} \cdot err_P(h^y) \\ &\leq \quad (1 + \epsilon) \cdot err_P(h^y) \end{aligned}$$

where the last inequality used the fact that $\frac{1+\epsilon/4}{1-\epsilon/4} \leq 1 + \epsilon$ for all $\epsilon \in (0, 1]$.

3.2 An Algorithmic Framework

This subsection will describe a strategy to obtain a function f satisfying (8)-(10). Later, Section 3.4 will instantiate the framework into a concrete algorithm by resorting to random sampling. Our discussion assumes that $|P| \geq 8$. Otherwise, we probe the entire P and return $f(h^\tau) = err_P(h^\tau)$ for all $\tau \in \mathbb{R}$.

Our framework demands a function

$$g_1 : \mathbb{H}_{mono} \to \mathbb{R}$$

that approximates err_P up to absolute error $\epsilon|P|/256$, namely,

$$|g_1(h^\tau) - err_P(h^\tau)| \quad \leq \quad \epsilon |P|/256 \tag{11}$$

for any h^τ. We defer the construction of g_1 to Section 3.4.

Define:

- α = the smallest $\tau \in \mathbb{R}$ with $g_1(h^\tau) < |P| \cdot (\frac{1}{4} - \frac{\epsilon}{256})$;
- β = the largest $\tau \in \mathbb{R}$ with $g_1(h^\tau) < |P| \cdot (\frac{1}{4} - \frac{\epsilon}{256})$.

When α and β do not exist. This happens because $g_1(h^\tau) \geq |P|(\frac{1}{4} - \frac{\epsilon}{256})$ for all $\tau \in \mathbb{R}$. We simply return

$$f(h^\tau) \quad = \quad g_1(h^\tau). \tag{12}$$

When α and β exist. By definition, it holds that $\alpha \leq \beta$. Obtain

$$P' \quad = \quad P \cap [\alpha, \beta]. \tag{13}$$

Lemma 10. *When $|P| \geq 8$, $|P'| \leq \frac{5}{8}|P|$.*

Proof. P' must have less than $|P|/4$ points of label 1. Otherwise, P has at least $|P|/4$ label-1 points in $[\alpha, \beta]$, which indicates $err_P(h^\beta) \geq |P|/4$. By (11), we ought to have $g_1(h^\beta) \geq |P| \cdot (\frac{1}{4} - \frac{\epsilon}{256})$, contradicting the definition of β. Similarly, $P' \setminus \{\alpha\}$ must have less than $|P|/4$ points of label 0. Otherwise, $err_P(h^\alpha) \geq |P|/4$, leading to $g_1(h^\alpha) \geq |P| \cdot (\frac{1}{4} - \frac{\epsilon}{256})$, also a contradiction. Therefore, the size of P' is less than $|P|/4 + |P|/4 + 1 = |P|/2 + 1$, where the "+1" counts point α itself. When $|P| \geq 8$, $|P|/2 + 1 \leq \frac{5}{8}|P|$. □

Our framework demands another function

$$g_2 : \mathbb{H}_{mono} \to \mathbb{R}$$

satisfying two requirements:

(1) g_2 approximates $err_{P \setminus P'}$ up to absolute error $\epsilon|P|/256$, namely, it holds for any τ

$$|g_2(h^\tau) - err_{P \setminus P'}(h^\tau)| \quad \leq \quad \epsilon |P|/256; \tag{14}$$

(2) for any $\tau \in [\alpha, \beta]$, $g_2(h^\tau) = g_2(h^\beta)$.

We defer the construction of g_2 to Section 3.4.

From P', we recursively obtain a function $f' : \mathbb{H}_{mono} \to [0, \infty)$ such that for any $h^\tau \in \mathbb{H}_{mono}$:

$$|f'(h^\tau) - err_{P'}(h^\tau)| \quad \leq \quad \epsilon |P'|/64 \tag{15}$$

and

$$err_{P'}(h^\tau) \cdot \left(1 - \frac{\epsilon}{4}\right) + \Delta' \leq f'(h^\tau) \leq err_{P'}(h^\tau) \cdot \left(1 + \frac{\epsilon}{4}\right) + \Delta' \tag{16}$$

where Δ' is an (unknown) real value satisfying

$$|\Delta'| \leq \epsilon|P'|/16. \tag{17}$$

After this, the desired function f is finalized as:

$$f(h^\tau) \quad = \quad g_2(h^\tau) + f'(h^\tau). \tag{18}$$

3.3 Correctness of the Framework

This subsection will prove that our framework always produces a function f obeying (8)-(10). This is obvious when $|P| \leq 7$. The subsequent analysis assumes $|P| \geq 8$.

3.3.1 When α and β Do Not Exist. In this scenario, the constructed f is given in (12). We will show that f satisfies (8)-(9) with $\Delta = 0$ (which trivially obeys (10)).

Proof of (8)**.** This directly follows from (11) and the fact that $f(h^\tau) = g_1(h^\tau)$.

Proof of (9)**.** The absence of α and β indicates that $g_1(h^\tau) \geq |P|(\frac{1}{4} - \frac{\epsilon}{256})$ for all $\tau \in \mathbb{R}$. Equipped with this, we can derive from (11):

$$err_P(h^\tau) \geq g_1(h^\tau) - \frac{\epsilon|P|}{256} \geq \frac{|P|}{4} - \frac{\epsilon|P|}{256} - \frac{\epsilon|P|}{256} \geq \frac{31|P|}{128} \tag{19}$$

Applying (11) again yields:

$$\begin{aligned} |g_1(h^\tau) - err_P(h^\tau)| \quad &\leq \quad \frac{\epsilon|P|}{256} \\ \text{(by (19))} \quad &\leq \quad \frac{\epsilon}{256} \cdot \frac{128 \cdot err_P(h^\tau)}{31} < \frac{\epsilon \cdot err_P(h^\tau)}{4}. \end{aligned}$$

Hence, the above indicates

$$err_P(h^\tau) \cdot \left(1 - \frac{\epsilon}{4}\right) \leq f(h^\tau) \leq err_P(h^\tau) \cdot \left(1 + \frac{\epsilon}{4}\right)$$

which satisfies (9) with $\Delta = 0$.

3.3.2 When α and β Exist. Inductively, assuming that (15)-(17) hold on P', we will show that the function f produced in (18) satisfies (8)-(10) with

$$\Delta \quad = \quad \Delta' + g_2(h^\beta) - err_{P\setminus P'}(h^\beta). \tag{20}$$

Proof of (8)**.** For any $h^\tau \in \mathbb{H}_{mono}$, it holds that:

$$err_P(h^\tau) \quad = \quad err_{P'}(h^\tau) + err_{P\setminus P'}(h^\tau). \tag{21}$$

Equipped with the above and (18), we have:

$$\begin{aligned} |f(h^\tau) - err_P(h^\tau)| \quad &\leq \quad |f'(h^\tau) - err_{P'}(h^\tau)| + |g_2(h^\tau) - err_{P\setminus P'}(h^\tau)| \\ \text{(by (14) and (15))} \quad &\leq \quad \frac{\epsilon|P'|}{64} + \frac{\epsilon|P|}{256} \\ \text{(by Lemma 10)} \quad &\leq \quad \frac{\epsilon|P|}{64} \cdot \frac{5}{8} + \frac{\epsilon|P|}{256} < \frac{\epsilon|P|}{64}. \end{aligned}$$

Proof of (10)**.** From (14), (17), and (20), we know

$$\begin{aligned} |\Delta| = |\Delta' + g_2(h^\beta) - err_{P\setminus P'}(h^\beta)| \quad &\leq \quad \frac{\epsilon|P'|}{16} + \frac{\epsilon|P|}{256} \\ \text{(by Lemma 10)} \quad &\leq \quad \frac{\epsilon|P|}{16} \cdot \frac{5}{8} + \frac{\epsilon|P|}{256} \\ &= \quad \frac{11\epsilon|P|}{256} \\ &< \quad \epsilon|P|/16. \end{aligned} \tag{22}$$

Proof of (9)**.** We will prove that $f(h^\tau)$ satisfies (9) for all $\tau \in \mathbb{R}$ by combining the next two lemmas.

Lemma 11. (9) *holds when $\tau < \alpha$ or $\tau > \beta$.*

Proof. By the definitions of α and β, when $\tau < \alpha$ or $\tau > \beta$, we have $g_1(h^\tau) \geq |P|(\frac{1}{4} - \frac{\epsilon}{256})$. Therefore, the same derivation in (19) also applies to $err_P(h^\tau)$.

We will first prove

$$f(h^\tau) \leq err_P(h^\tau)(1 + \epsilon/4) + \Delta. \tag{23}$$

Combining (19) and (22) leads to

$$\frac{\epsilon \cdot err_P(h^\tau)}{4} + \Delta \quad \geq \quad \frac{31\epsilon|P|}{128 \cdot 4} - \frac{11\epsilon|P|}{256} = \frac{9\epsilon|P|}{512}. \tag{24}$$

As proved earlier, f satisfies (8). Hence, we have:

$$\begin{aligned} f(h^\tau) - err_P(h^\tau) \quad &\leq \quad \frac{\epsilon|P|}{64} < \frac{9\epsilon|P|}{512} \\ \text{(applying (24))} \quad &\leq \quad \frac{\epsilon \cdot err_P(h^\tau)}{4} + \Delta \end{aligned} \tag{25}$$

which gives (23).

Next, we will prove

$$f(h^\tau) \geq err_P(h^\tau)(1 - \epsilon/4) + \Delta. \tag{26}$$

From (19) and (22), we have

$$\frac{\epsilon \cdot err_P(h^\tau)}{4} - \Delta \quad \geq \quad \frac{31\epsilon|P|}{512} - \frac{11\epsilon|P|}{256} = \frac{9\epsilon|P|}{512}.$$

Therefore, (8) tells us

$$err_P(h^\tau) - f(h^\tau) \leq \frac{\epsilon|P|}{64} < \frac{9\epsilon|P|}{512} \leq \frac{\epsilon \cdot err_P(h^\tau)}{4} - \Delta$$

which gives (26). □

Lemma 12. (9) *holds when $\alpha \leq \tau \leq \beta$.*

Proof. For any $\tau \in [\alpha, \beta]$, we must have

$$err_{P\setminus P'}(h^\tau) \quad = \quad err_{P\setminus P'}(h^\beta). \tag{27}$$

The second requirement of g_2 ensures that $g_2(h^\tau) = g_2(h^\beta)$ for all such τ. This, together with (18), yields $f(h^\tau) = g_2(h^\beta) + f'(h^\tau)$.

We can thus derive:

$$\begin{aligned} & err_P(h^\tau)(1 + \epsilon/4) + \Delta \\ \text{(by (21) and (27))} \quad = \quad & \left(err_{P'}(h^\tau) + err_{P\setminus P'}(h^\beta)\right)(1 + \epsilon/4) + \Delta \\ \geq \quad & err_{P'}(h^\tau)(1 + \epsilon/4) + err_{P\setminus P'}(h^\beta) + \Delta \\ \text{(by (20))} \quad = \quad & err_{P'}(h^\tau)(1 + \epsilon/4) + \Delta' + g_2(h^\beta) \\ \text{(by (16))} \quad \geq \quad & f'(h^\tau) + g_2(h^\beta) \\ = \quad & f(h^\tau). \end{aligned}$$

Similarly:

$$\begin{aligned}
& err_P(h^\tau)(1-\epsilon/4)+\Delta \\
= \;& \left(err_{P'}(h^\tau)+err_{P\setminus P'}(h^\beta)\right)(1-\epsilon/4)+\Delta \\
\leq \;& err_{P'}(h^\tau)(1-\epsilon/4)+err_{P\setminus P'}(h^\beta)+\Delta \\
= \;& err_{P'}(h^\tau)(1-\epsilon/4)+\Delta'+g_2(h^\beta) \\
\text{(by (16))} \;\leq \;& f'(h^\tau)+g_2(h^\beta) \\
= \;& f(h^\tau).
\end{aligned}$$

This establishes (9). □

3.4 A Concrete Algorithm

To instantiate our framework of Section 3.2 into an actual algorithm, we need to explain how to obtain the required functions g_1 and g_2. The next discussion sets h to the number of levels in the recursion; clearly, $h = O(\log n)$ due to Lemma 10.

Obtaining g_1. Take with replacement a uniform sample S_1 of P with size $|S_1| = O(\frac{1}{\epsilon^2}\log\frac{|P|h}{\delta})$, and define

$$g_1(h^\tau) \;=\; \frac{|P|}{|S_1|}\cdot err_{S_1}(h^\tau).$$

The discussion in Section 2 indicates that, for one $h^\tau \in \mathbb{H}_{mono}(P)$, $g_1(h^\tau)$ satisfies (11) with probability at least $1-\frac{\delta}{2h(|P|+1)}$. Since $|\mathbb{H}_{mono}(P)| = |P|+1$, g_1 satisfies (11) on the whole $\mathbb{H}_{mono}(P)$ (and hence, the whole $\mathbb{H}_{mono}$) with probability at least $1-\delta/(2h)$.

Obtaining g_2. After P' has been obtained (using g_1), take with replacement a uniform sample S_2 of $P\setminus P'$ with size $|S_2| = O(\frac{1}{\epsilon^2}\log\frac{|P\setminus P'|h}{\delta}) = O(\frac{1}{\epsilon^2}\log\frac{|P|h}{\delta})$, and define

$$g_2(h^\tau) \;=\; \frac{|P\setminus P'|}{|S_2|}\cdot err_{S_2}(h^\tau). \tag{28}$$

An argument analogous to the one for g_1 shows that g_2 satisfies (14) on the whole $\mathbb{H}_{mono}$ with probability at least $1-\delta/(2h)$. Furthermore, g_2 satisfies the second requirement prescribed in Section 3.2 because S_2 contains no points in $[\alpha,\beta]$.

Total cost. In summary, by probing $O(\frac{1}{\epsilon^2}\log\frac{|P|h}{\delta})$ points at each level of the recursion, we can find the desired functions g_1 and g_2 with probability at least $1-\delta/h$. As there are h levels, the overall probing cost is $O(\frac{h}{\epsilon^2}\log\frac{n}{\delta}) = O(\frac{\log n}{\epsilon^2}\cdot\log\frac{n}{\delta})$, and the success probability is at least $1-\delta$. It is rudimentary to implement our algorithm in $\tilde{O}(1/\epsilon^2)$ time (by creating augmented binary search trees[2] on the sample points). This completes the proof of Lemma 9.

3.5 A Weighted View

We close this section by providing a "weighted view" of our algorithm, which will become useful in the later sections.

Consider our algorithm (combining Sections 3.2 and 3.4) again.

- When $|P| \geq 8$, for each level of recursion, define a *fully-labeled weighted sample* as follows:
  - If α and β do not exist, the fully-labeled weighted sample is the set S_1 explained in Section 3.4, with every point $p \in S_1$ assigned a weight of $|P|/|S_1|$.
  - Otherwise, the fully-labeled weighted sample is S_2 (again, see Section 3.4), with every point $p \in S_2$ assigned a weight of $|P\setminus P'|/|S_2|$.

  In either case, our algorithm probes all the points in the fully-labeled weighted sample. It is easy to verify that, if α and β do not exist, $g_1(h^\tau) = w\text{-}err_{S_1}(h^\tau)$ (see (3) for the definition of w-err); otherwise, $g_2(h^\tau) = w\text{-}err_{S_2}(h^\tau)$.
- When $|P| \leq 7$, also define a full-labeled weighted sample, which is simply P, with every point assigned a weight of 1.

Thus, each recursion level produces a fully-labeled weighted sample. Define:

Σ = the union of all the fully-labeled weighted samples.

Our algorithm returns Σ as a side product. The lemma below states an interesting fact:

Lemma 13. *The function f returned by our algorithm satisfies $f(h^\tau) = w\text{-}err_\Sigma(h^\tau)$ for any $h^\tau \in \mathbb{H}_{mono}$.*

Proof. The claim obviously holds when $|P| \leq 7$. Assuming that the claim is true for all P with at least $t \geq 7$ points, next we prove the correctness for $|P| = t+1$.

When α and β do not exist, $f(h^\tau) = g_1(h^\tau)$ and no recursion occurs. In this case, Σ is simply the set S_1 in Section 3.4. Hence, $f(h^\tau) = g_1(h^\tau) = w\text{-}err_{S_1}(h^\tau) = w\text{-}err_\Sigma(h^\tau)$.

When α and β exist, the algorithm recursively processes P' (see (13)) whose size is strictly smaller than $|P|$ (Lemma 10). Suppose that the recursion returns a function f' and a fully-labeled weighted set Σ'. By the inductive assumption, $f'(h^\tau) = w\text{-}err_{\Sigma'}(h^\tau)$. The final fully-labeled weighted set returned for P is $\Sigma = S_2 \cup \Sigma'$. Thus, $f(h^\tau) = g_2(h^\tau)+f'(h^\tau) = w\text{-}err_{S_2}(h^\tau)+w\text{-}err_{\Sigma'}(h^\tau) = w\text{-}err_\Sigma(h^\tau)$. □

4 ACTIVE CLASSIFICATION IN MULTI-DIMENSIONAL SPACE

This section will extend our 1D algorithm to solve the active monotone classification problem in $\mathbb{R}^d$ for arbitrary $d \geq 1$ and $\epsilon > 0$, and prove Theorems 2 and 3.

4.1 The Algorithm

This subsection serves as a proof of:

Lemma 14. *By probing $O((w/\epsilon^2)\cdot\log n\cdot\log(n/w))$ points in P, with probability at least $1-1/n^2$ we can obtain a fully-labeled weighted set Σ of size $O((w/\epsilon^2)\cdot\log n\cdot\log(n/w))$ having the property below for any classifiers h and h' in $\mathbb{H}_{mono}$:*

$$w\text{-}err_\Sigma(h) \leq w\text{-}err_\Sigma(h') \text{ implies } err_P(h) \leq_\epsilon err_P(h')$$

where w-err is defined in (3) and $\leq_\epsilon$ is defined in Section 1.1. Our algorithm runs in $\tilde{O}(dn+n^{2.5}+w/\epsilon^2)$ time, where $\tilde{O}(.)$ hides a factor polylogarithmic to d, n, and $1/\epsilon$.

[2] See Section 14.1 of [8].

As before, denote by n and w the size and the dominance width of the input set P, respectively (see Section 2 for the definition of w). We start by using Lemma 6 to obtain a chain decomposition of P with precisely w chains: $C_1, C_2, ..., C_w$.

For every $i \in [1, w]$, we will compute a fully-labeled weighted set Σ_i, which is a subset of C_i ensuring:

$$err_{C_i}(h) \cdot \left(1 - \frac{\epsilon}{4}\right) - \Delta_i \leq w\text{-}err_{\Sigma_i}(h) \leq err_{C_i}(h) \cdot \left(1 - \frac{\epsilon}{4}\right) + \Delta_i \quad (29)$$

for every $h \in \mathbb{H}_{mono}$, where Δ_i is some fixed real value unknown to our algorithm. Once this is done, we obtain

$$\Sigma \quad = \quad \bigcup_{i=1}^{w} \Sigma_i. \quad (30)$$

It must hold that

$$err_P(h) \cdot \left(1 - \frac{\epsilon}{4}\right) - \Delta \leq w\text{-}err_{\Sigma}(h) \leq err_P(h) \cdot \left(1 - \frac{\epsilon}{4}\right) + \Delta \quad (31)$$

where Δ equals $\sum_{i=1}^{w} \Delta_i$ and remains unknown. Indeed, (31) follows from (29), plugging in $err_P(h) = \sum_{i=1}^{w} err_{C_i}(h)$ and $w\text{-}err_{\Sigma}(h) = \sum_{i=1}^{w} w\text{-}err_{\Sigma_i}(h)$. By the same argument in Section 3.1 (about how (9) leads to the ϵ-comparison property), we can show that the property in Lemma 14 is a corollary of (31).

Next, we will concentrate on finding Σ_i for chain C_i. Observe that this is in fact a 1D problem. To explain, let us sort the points of C_i according to $\succcurlyeq$, i.e., each point p in the sorted C_i dominates all the points after p. *Every* classifier $h \in \mathbb{H}_{mono}$ maps only a *prefix* of the points in the sorted C_i to 1 and, therefore, can be regarded as a 1D classifier on C_i of the form (6). By combining Lemmas 9 and 13, we conclude that we can produce the desired Σ_i with probability at least $1 - \frac{1}{w \cdot n^2}$ by probing $O(\frac{1}{\epsilon^2} \cdot \log |C_i| \cdot \log(wn^2)) = O(\frac{\log n}{\epsilon^2} \log |C_i|)$ points in C_i.

Considering all the w chains $C_1, ..., C_w$, we now claim that the target $\Sigma_1, ..., \Sigma_w$ can be produced with probability at least $1 - 1/n^2$ with a total probing cost of

$$O\left(\frac{\log n}{\epsilon^2} \sum_{i=1}^{w} \log |C_i|\right) \quad = \quad O\left(\frac{\log n}{\epsilon^2} \cdot w \log \frac{n}{w}\right)$$

where the inequality used the fact that $\sum_{i=1}^{w} \log |C_i|$ is maximized when all the chains have the same size. The same bound also applies to the size of Σ (see (30)).

Finally, let us analyze the CPU time. Lemma 6 shows that finding the chain decomposition takes $O(dn + n^{2.5})$ time. By Lemma 9, the computation of each Σ_i requires $\tilde{O}(1/\epsilon^2)$ time; hence, the production of all $\Sigma_1, ..., \Sigma_w$ incurs $\tilde{O}(w/\epsilon^2)$ time. This completes the proof of Lemma 14.

4.2 Proof of Theorem 2

Since *all* the points in the Σ of Lemma 14 have their labels revealed, we can identify the classifier $h^{\#} \in \mathbb{H}_{mono}$ having the smallest $w\text{-}err_{\Sigma}(h^{\#})$ with no more probing (CPU time is not a concern here). The property in Lemma 14 guarantees that $err_P(h^{\#}) \leq (1 + \epsilon)k^*$, where k^* is the error on P of the optimal monotone classifier. This establishes Theorem 2.

4.3 Proof of Theorem 3

Identifying the classifier $h^{\#}$ mentioned in Section 4.2 is an instance of Problem 2 (passive weighted monotone classification). Provided that we can solve the instance in $T_{prob2}(d, N)$ time where $N = O((w/\epsilon^2) \cdot \log n \cdot \log(n/w))$, our algorithm for Problem 1 runs in $\tilde{O}(dn^2 + n^{2.5} + w/\epsilon^2) + T_{prob2}(d, N)$ time overall. This completes the proof of Theorem 3.

The next section will prove that $T_{prob2}(d, N) = \tilde{O}(dN^2 + N^3)$.

5 PASSIVE MONOTONE CLASSIFICATION

Now that we have already established Theorems 2 and 3, the last missing piece in our overall solution to Problem 1 is an algorithm for solving Problem 2 (passive weighted monotone classification) efficiently. This section will complete the piece by proving Theorem 4.

5.1 The Algorithm

Recall that the input to Problem 2 is a fully-labeled weighted set P of n points in $\mathbb{R}^d$. We say that a point $p \in P$ is *contending* in either situation below:

- $label(p) = 0$ but there is a label-1 point $q \in P$ such that $p \succcurlyeq q$;
- $label(p) = 1$ but there is a label-0 point $q \in P$ such that $q \succcurlyeq p$.

Define

$$P^{con} \quad = \quad \{p \in P \mid p \text{ is contending}\}.$$

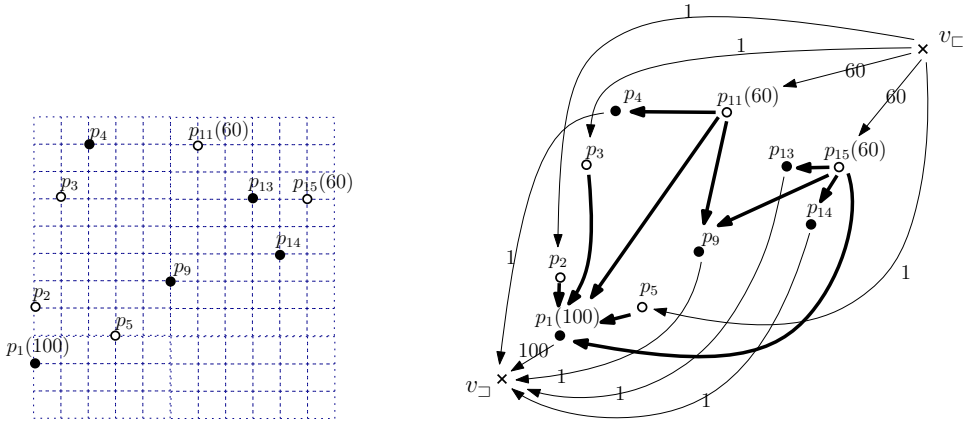
As an example, Figure 2(a) shows all the contending points in Figure 1(b).

Lemma 15. *If there is a monotone classifier h whose weighted error on P^{con} is k, then there is a monotone classifier whose weighted error on P is k.*

Proof. First, construct a function h' as follows: (i) $h'(p) = h(p)$ for every $p \in P^{con}$, and (ii) $h'(p) = label(p)$ for every $p \notin P^{con}$. We claim that there do not exist $p, q \in P$ such that $p \succcurlyeq q$, but $h'(p) = 0$ and $h'(q) = 1$. The claim implies the correctness of the lemma.

Suppose, on the contrary, that such p, q exist. They cannot both belong to P^{con} due to the monotonicity of h'. Assume that $p \notin P^{con}$, which means $label(p) = h'(p) = 0$. We can then assert that $label(q) = 0$ (otherwise, p would be contending). The fact $h'(q) \neq label(q)$ indicates that q must be contending. Hence, there must be a label-1 point q' satisfying $q \succcurlyeq q'$. Because $p \succcurlyeq q'$, the presence of q' violates the assumption that $p \notin P^{con}$. A symmetric argument shows that $q \notin P^{con}$.

Thus, we know that neither p nor q can be contending. Hence, $label(p) = h'(p) = 0$ and $label(q) = h'(q) = 1$, but this is impossible when p and q are non-contending. We now conclude that such p and q cannot exist. □



(a) Contending points in Figure 1(b)

(b) The constructed max-flow problem (the bold edges have capacity ∞)

Figure 2: Illustration of our algorithm for Problem 2

Lemma 15 indicates that it suffices to solve the passive weighted monotone classification problem on P^{con}.[3] We partition P^{con} into:

$$P_0^{con} = \{p \in P^{con} \mid label(p) = 0\}$$
$$P_1^{con} = \{q \in P^{con} \mid label(q) = 1\}.$$

Construct a graph $G = (V, E)$ for the max-flow problem (Section 2) as follows:

- $V = P^{con} \cup \{v_\sqsubset, v_\sqcap\}$ where $v_\sqsubset$ and $v_\sqcap$ are special vertices being the *source* and *sink*, respectively.
- **(Edge type 1)** For each point $p \in P_0^{con}$, add to E an edge $e = (v_\sqsubset, p)$ with $cap(e) = weight(p)$.
- **(Edge type 2)** For each point $q \in P_1^{con}$, add to E an edge $e = (q, v_\sqcap)$ with $cap(e) = weight(q)$.
- **(Edge type 3)** For each $(p, q) \in P_0^{con} \times P_1^{con}$ such that $p \succcurlyeq q$, add to E an edge $e = (p, q)$ with $cap(e) = \infty$.

Notice that all edges in E have positive capacities. Furthermore, every path from $v_\sqsubset$ to $v_\sqcap$ must have the form $v_\sqsubset \to p \to q \to v_\sqcap$, where $p \in P_0^{con}$ and $q \in P_1^{con}$.

Figure 2(b) demonstrates the graph G constructed for the set P^{con} of points in Figure 2(a). There are five type-1 edges: from $v_\sqsubset$ to p_2, p_3, p_5, p_{11}, and p_{15} with weights 1, 1, 1, 60, and 60, respectively. There are five type-1 edges: from p_1, p_4, p_9, p_{13}, p_{14} to $v_\sqcap$ with weights 100, 1, 1, 1, and 1, respectively. The other edges, shown in bold, are of type 3.

Now, run a max-flow algorithm to obtain a cut-edge set E^*_{cut} with the minimum weight (see the discussion in Section 2 for why this is possible). Then, construct a classifier h^*_{cut} based on E^*_{cut}:

- For each point $p \in P_0^{con}$, $h^*_{cut}(p) = 1$ if $(v_\sqsubset, p) \in E^*_{cut}$; otherwise, $h^*_{cut}(p) = 0$.
- For each point $q \in P_1^{con}$, $h^*_{cut}(q) = 0$ if $(q, v_\sqcap) \in E^*_{cut}$; otherwise, $h^*_{cut}(p) = 1$.

In the example of Figure 2(b), an optimal E^*_{cut} includes 5 edges: $(p_1, v_\sqcap)$, $(p_4, v_\sqcap)$, $(p_9, v_\sqcap)$, $(p_{13}, v_\sqcap)$, and $(p_{14}, v_\sqcap)$. Note that the weight of E^*_{cut}, as defined in (5), equals 104. Accordingly, the constructed h^*_{cut} maps all the points in Figure 2(b) to 0.

The next two lemmas, which we will prove in the subsequent sections, indicate that h^*_{cut} is exactly what we are looking for.

Lemma 16. *These do not exist $p, q \in P^{con}$ such that $p \succcurlyeq q$ but $h^*_{cut}(p) = 0$ and $h^*_{cut}(q) = 1$.*

Lemma 17. *The weight of h^*_{cut} on P_{con} equals the smallest weighted error on P_{con} of all monotone classifiers.*

Regarding the CPU time, the generation of G can be easily accomplished in $O(dn^2)$ time. We need to add the running time of the max-flow algorithm, which is $T_{maxflow}(n)$. Thus, subject to the correctness of Lemmas 16 and 17, we complete the proof of Theorem 4.

5.2 Proof of Lemma 16

Let us start with a property of E^*_{cut}:

Lemma 18. *E^*_{cut} does not contain any edge of type 3.*

Proof. All the edges of type 3 have an infinite capacity, while the other edges have finite capacities. Suppose that E^*_{cut} has an edge (p, q) of type 3. Then, we can obtain another cut-edge set of lower weight by removing (p, q) and adding all the incoming edges of p. □

Henceforth, denote by $\overline{G}$ the graph induced by the edges in $E \setminus E_{cut}$; the definition of E_{cut} indicates that $\overline{G}$ should contain no paths from $v_\sqsubset$ to $v_\sqcap$. Suppose that Lemma 16 is incorrect, i.e., points p and q in the lemma's statement actually exist. We will prove a contradiction in each of the following 4 cases.

[3] If the optimal weighted error on P is k, then obviously the optimal weighted error on P^{con} is at most k. Combining this with Lemma 15 shows that P and P^{con} must have the same optimal weighted error.

Case 1: $label(p) = 0$ and $label(q) = 1$. This means that h^*_{cut} correctly classifies both p and q. Hence, neither $(v_\sqsubset, p)$ nor $(q, v_\sqsupset)$ is in E^*_{cut}. However, since (p, q) is not in E^*_{cut} either (Lemma 18), we have found a path $v_\sqsubset \rightarrow p \rightarrow q \rightarrow v_\sqsupset$ in $\overline{G}$, giving a contradiction.

Case 2: $label(p) = 0$ and $label(q) = 0$. This means that h^*_{cut} correctly classifies p but mis-classifies q. Therefore, $(v_\sqsubset, p) \notin E^*_{cut}$ but $(v_\sqsubset, q) \in E^*_{cut}$. Now, construct

$$E_{cut} \quad = \quad E^*_{cut} \setminus \{(v_\sqsubset, q)\}.$$

We argue that E_{cut} is a still a cut-edge set, thus contradicting the fact that E^*_{cut} is a cut-edge set of the minimum weight (recall that all edges in E have positive capacities).

Suppose that E_{cut} is not a cut-edge set. This indicates the existence of a path π from $v_\sqsubset$ to $v_\sqsupset$ after removing all the edges in E_{cut}. As π does not exist in $\overline{G}$, π must definitely use the edge $(v_\sqsubset, q)$. Without loss of generality, write π as $v_\sqsubset \rightarrow q \rightarrow q' \rightarrow v_\sqsupset$, for some $q' \in P^{con}_1$. This means that the edge $(q', v_\sqsupset)$ does not belong to E^*_{cut}.

The existence of edge (q, q') implies that $q \succcurlyeq q'$. Therefore, $p \succcurlyeq q \succcurlyeq q'$, which means that the edge (p, q') must be an edge in E. As (p', q) cannot belong to E^*_{cut} (Lemma 18), we have found a path $v_\sqsubset \rightarrow p \rightarrow q' \rightarrow v_\sqsupset$ in $\overline{G}$, giving a contradiction.

Case 3: $label(p) = 1$ and $label(q) = 1$. This case can be dealt with using an argument symmetric to the one for Case 2.

Case 4: $label(p) = 1$ and $label(q) = 0$. This means that h^*_{cut} misclassifies both p and q. Therefore, both $(v_\sqsubset, q)$ and $(p, v_\sqsupset)$ belong to E^*_{cut}.

We claim:

> *every* point $q' \in P^{con}_1$ with $q \succcurlyeq q'$ must have its edge $(q', v_\sqsupset)$ included in E^*_{cut}.

Otherwise, i.e., $(q', v_\sqsupset) \notin E^*_{cut}$, h^*_{cut} correctly classifies q', i.e., $h^*_{cut}(q') = 1$. But then we have found p, q' such that $p \succcurlyeq q'$, $h^*_{cut}(p) = 0$, $h^*_{cut}(q') = 1$, $label(p) = 1$, and $label(q') = 1$. This is a Case-3 scenario, which has been shown to be impossible.

Now, construct

$$E_{cut} \quad = \quad E^*_{cut} \setminus \{(v_\sqsubset, q)\}.$$

We argue that E_{cut} is a still a cut-edge set, thus contradicting the definition of E^*_{cut}.

Suppose that E_{cut} is not a cut-edge set. Hence, a path π exists from $v_\sqsubset$ to $v_\sqsupset$ after removing the edges in E_{cut}. As π is absent in $\overline{G}$, π must use the edge $(v_\sqsubset, q)$. Without loss of generality, write π as $v_\sqsubset \rightarrow q \rightarrow q' \rightarrow v_\sqsupset$, for some $q' \in P^{con}_1$. This means that the edge $(q', v_\sqsupset)$ does not belong to E^*_{cut}. However, this violates our earlier claim that $(q', v_\sqsupset)$ must appear in E^*_{cut}.

5.3 Proof of Lemma 17

The classifier h^*_{cut} constructed by our algorithm mis-classifies (i) a point $p \in P^{con}_0$ if and only if $(v_\sqsubset, p) \in E^*_{cut}$ and (ii) a point $p \in P^{con}_1$ if and only if $(p, v_\sqsupset) \in E^*_{cut}$. Therefore, the weighted error of h^*_{cut} on P^{con} is precisely the weight of E^*_{cut}, that is, $w\text{-}err_{P^{con}}(h^*_{cut}) = weight(E^*_{cut})$.

Suppose that Lemma 17 is wrong such that some monotone classifier h has a weighted error on P^{con} strictly smaller than h^*_{cut}. Next, we will construct a cut-edge set E_{cut} of G whose weight is strictly less than that of E^*_{cut}, thus contradicting the optimality of E^*_{cut}.

Specifically, for every point $p \in P^{con}$ mis-classified by h, we add to E_{cut} an edge

- $(v_\sqsubset, p)$ if $label(p) = 0$;
- $(p, v_\sqsubset)$ otherwise.

Next, we argue that E_{cut} is indeed a cut-edge set of G. As it is easy to verify that $w\text{-}err_{P^{con}}(h) = weight(E_{cut})$, we have $weight(E_{cut}) < weight(E^*_{cut})$, giving the desired contradiction.

Suppose that a path exists from $v_\sqsubset$ to $v_\sqsupset$ after removing all the edges in E_{cut} from G. Let the path be $v_\sqsubset \rightarrow p \rightarrow q \rightarrow v_\sqsupset$ for some $p \in P^{con}_0$ and $q \in P^{con}_1$. As neither $(v_\sqsubset, p)$ nor $(q, v_\sqsupset)$ belongs to E_{cut}, h must classify both p and q correctly, i.e., $h(0) = 0$ and $h(q) = 1$. This, however, violates the monotonicity of h because $p \succcurlyeq q$.

6 OPTIMAL ACTIVE MONOTONE CLASSIFICATION NEEDS $\Omega(n)$ PROBES

This section will establish Theorem 1 by proving a lower bound for Problem 1. Henceforth, the dimensionality d will be fixed to 1; thus, each "point" is merely a real value. The value of ϵ will be fixed to 0 (the goal is to find an optimal classifier).

6.1 The Proof Framework

Our proof will assume that the input size n is an even number. Let us define a family $\mathcal{P}$ of n inputs, which are based on the same set of points $\{1, 2, ..., n\}$, but differ in the design of labels. By default, assign label 1 (or 0) to each odd (or even, resp.) number in $[1, n]$. Then, for every integer $i \in [1, n/2]$, define two inputs:

- $P_{00}(i)$, where all points take the default lablels except $2i - 1$, which is assigned label 0;
- $P_{11}(i)$, where all points take the default lablels except $2i$, which is assigned label 1.

The constructed $\mathcal{P}$ is $\{P_{00}(1), P_{00}(2), ..., P_{00}(n/2), P_{11}(1), P_{11}(2), ..., P_{11}(n/2)\}$. Every $P_{00}(i)$ or $P_{11}(i)$ will be referred to as a *00-input* or *11-input*, respectively.

We can understand the inputs in $\mathcal{P}$ in a different manner. Chop the points $1, 2, ..., n$ into $n/2$ pairs $(1, 2), (3, 4), ..., (n - 1, n)$. In a *normal pair* $(x - 1, x)$, points $x - 1$ and x carry labels 1 and 0, respectively. Each input $P \in \mathcal{P}$ contains exactly one *anomaly pair* $(x - 1, x)$. If P is a 00-input, both $x - 1$ and x are assigned label 0; otherwise, they are assigned label 1.

For each $P \in \mathcal{P}$, an optimal monotone classifier has an error of $n/2 - 1$. Indeed, every monotone classifier must mis-classify at least one point of each normal pair in P. On the other hand, we can achieve the error of $n/2 - 1$ by mapping all the points to 1 (for a 11-input) or 0 (for a 00-input).

Fix an arbitrary deterministic algorithm A designed for Problem 1. We say that A *errs* on $P \in \mathcal{P}$ if the output classifier of A is non-optimal for P. Denote by $cost_P(A)$ the number of probes performed by A when executed on P. Define:

$$\begin{aligned} nonoptcnt(A) &= \sum_{P \in \mathcal{P}} \mathbb{1}_{A \text{ errs on } P} \\ totalcost(A) &= \sum_{P \in \mathcal{P}} cost_P(A). \end{aligned}$$

Note that if A is randomized, then both $nonoptcnt(A)$ and $totalcost(A)$ are random variables. We will prove a crucial lemma in Section 6.2:

Lemma 19. *Fix any non-negative constant $c < 1$. When $n \geq \max\{4, 1/c\}$, the following holds for any deterministic algorithm A_{det}: if $nonoptcnt(A_{det}) \leq cn/2$, then $totalcost(A_{det}) = \Omega(n^2)$.*

The lemma leads to:

Corollary 20. *When $n \geq 4$, the following holds for any randomized algorithm A_{ran}: if $\mathbf{E}[nonoptcnt(A_{ran})] \leq n/3$, then $\mathbf{E}[totalcost(A_{ran})] = \Omega(n^2)$.*

The proof uses an argument in [25] and can be found in the appendix. The corollary implies Theorem 1. Indeed, if A_{ran} guarantees finding an optimal solution with probability at least $2/3$ on any input, then $\mathbf{E}[nonoptcnt(A_{ran})] \leq n/3$. Thus, the corollary tells us that, when n is sufficiently large, $\mathbf{E}[totalcost(A_{ran})] = \Omega(n^2)$. This means that the expected cost of A_{ran} is $\Omega(n)$ on at least one input in $\mathcal{P}$ because $\mathcal{P}$ has n inputs.

6.2 Proof of Lemma 19

We start by proving an important property of the family $\mathcal{P}$ constructed earlier:

Lemma 21. *No monotone classifier can be optimal for both $P_{00}(i)$ and $P_{11}(i)$ simultaneously.*

Proof. Recall that a 1D monotone classifier h has the form of (6), which takes a parameter $\tau \in P \cup \{-\infty\}$. We will show that no value of τ can make h optimal for both $P_{00}(i)$ and $P_{11}(i)$. As mentioned earlier, for each input, an optimal classifier should have an error of $n/2 - 1$.

- Case $\tau < 2i - 1$: on $P_{00}(i)$, h mis-classifies both $2i - 1$ and $2i$, and has a non-optimal error of $n/2 + 1$.
- Case $\tau = 2i - 1$: on $P_{00}(i)$, h mis-classifies $2i$, and has a non-optimal error of $n/2$.
- Case $\tau \geq 2i$: on $P_{11}(i)$, h mis-classifies both $2i - 1$ and $2i$, and has a non-optimal error of $n/2 + 1$.

The absence of τ implies the lemma's correctness. □

For the rest of the proof, we will increase the power of A_{det} in two ways. First, A_{det} knows that the input comes from the family $\mathcal{P}$. Second, we will give A_{det} some free labels. Specifically, every time A_{det} probes a point of some pair $(2i - 1, 2i)$ (where $1 \leq i \leq n/2$), we will reveal the label of the other point (in that pair) voluntarily. Next, by "A_{det} probing pair i", we mean that A_{det} has acquired the labels of both $2i - 1$ and $2i$. If Lemma 19 holds even on such an "empowered" A_{det}, it must hold on the original A_{det} because an empowered algorithm can choose to ignore the extra information.

We consider that A_{det} terminates immediately after probing an anomaly pair, i.e., catching the pair with the same label. Once the anomaly pair is found, there are no reasons to continue because A_{det} knows exactly the labels in all the normal pairs and, hence, can return an optimal classifier with full confidence.

With the above, we can model A_{det} as probing according to a predetermined sequence: pair x_1, pair x_2, ..., pair x_ℓ for some integer $\ell \in [0, n/2]$. Specifically, for each $j \in [1, \ell - 1]$, if pair x_j is an anomaly, the algorithm terminates; otherwise, it moves on to pair x_{j+1}. If all the ℓ pairs have been probed and no anomaly has been found, A_{det} outputs a fixed classifier h_{det}.

Consider running A_{det} on all the inputs in the family $\mathcal{P}$. For each

$$i \in \{1, 2, ..., n/2\} \setminus \{x_1, x_2, ..., x_\ell\} \tag{32}$$

A_{det} never probes pair i. Thus, the output of A_{det} must be h_{det} for both $P_{00}(i)$ and $P_{11}(i)$. By Lemma 21, we know that h_{det} cannot be optimal for both $P_{00}(i)$ and $P_{11}(i)$. Hence, A_{det} must err on either $P_{00}(i)$ or $P_{11}(i)$. This means that:

$$nonoptcnt(A_{det}) \geq n/2 - \ell. \tag{33}$$

To calculate $totalcost(A_{det})$, notice that A_{det} performs ℓ probes for $P_{00}(i)$ and $P_{11}(i)$ of every i satisfying (32), and $j \in [1, \ell]$ probes for $P_{00}(x_j)$ and $P_{11}(x_j)$. Hence:

$$\begin{aligned} totalcost(A_{det}) &= 2\ell \cdot (n/2 - \ell) + 2\sum_{j=1}^{\ell} j \\ &= n\ell - \ell^2 - \ell. \end{aligned} \tag{34}$$

For $nonoptcnt(A_{det})$ to be at most $cn/2$ (see the statement of Lemma 19), (33) tells us that $\ell \geq \frac{n}{2}(1 - c)$. Under such ℓ and the condition $n \geq 1/c$, (34) is at least

$$\frac{n^2}{4}(1 - c^2) - \frac{n}{2}(1 - c)$$

which is at least $n^2(1 - c^2)/8$ for $n \geq 4$. This completes the proof of Lemma 19.

7 CONCLUSIONS

Monotone classification is a fundamental topic at the core of many applications, especially those related to entity matching, record linkage, and duplicate detection. This paper has presented three main results that complement existing work towards building a systematic theory on the topic. The first two results concern *active* classification, where all the labels are hidden until probed, and the objective is to find an accurate monotone classifier with the least probes. We first prove that finding an optimal monotone classifier is hard, because the trivial solution of probing all labels is already asymptotically optimal. The hardness result officially opens the door to studying how to reduce the probing cost by aiming at near-optimal classifiers. For this purpose, we develop a new algorithm that returns (with high probability) a $(1 + \epsilon)$-approximate classifier and has a probing cost nearly matching a known lower bound. Our third result concerns *passive* active classification, where all the labels are directly given, and the objective is to discover an optimal

classifier with the least computation. We show that the problem can be settled in polynomial time.

ACKNOWLEDGMENTS

This work was partially supported by GRF grant 14207820 from HKRGC and a research grant from Alibaba Group.

REFERENCES

[1] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *Proceedings of ACM Management of Data (SIGMOD)*. 783–794.
[2] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. 2009. Agnostic active learning. *Journal of Computer and System Sciences (JCSS)* 75, 1 (2009), 78–89.
[3] Kedar Bellare, Suresh Iyengar, Aditya G. Parameswaran, and Vibhor Rastogi. 2013. Active Sampling for Entity Matching with Guarantees. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 12:1–12:24.
[4] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. 2009. Importance weighted active learning. In *Proceedings of International Conference on Machine Learning (ICML)*. 49–56.
[5] Guilherme Dal Bianco, Renata Galante, Marcos Andre Goncalves, Sergio D. Canuto, and Carlos Alberto Heuser. 2015. A Practical and Effective Sampling Selection Strategy for Large Scale Deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 27, 9 (2015), 2305–2319.
[6] Peter Christen, Dinusha Vatsalan, and Qing Wang. 2015. Efficient Entity Resolution with Adaptive and Interactive Training Data Selection. In *Proceedings of International Conference on Management of Data (ICDM)*. 727–732.
[7] Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. 2016. Distributed Data Deduplication. *Proceedings of the VLDB Endowment (PVLDB)* 9, 11 (2016), 864–875.
[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms, Second Edition.* The MIT Press.
[9] Sanjoy Dasgupta, Daniel J. Hsu, and Claire Monteleoni. 2007. A general agnostic active learning algorithm. In *Proceedings of Neural Information Processing Systems (NIPS)*. 353–360.
[10] Robert P. Dilworth. 1950. A Decomposition Theorem for Partially Ordered Sets. *The Annals of Mathematics* 51, 1 (1950), 161–166.
[11] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. 2017. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems* 65 (2017), 137–157.
[12] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. 2014. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of ACM Management of Data (SIGMOD)*. 601–612.
[13] Andrew V. Goldberg and Satish Rao. 1998. Beyond the Flow Decomposition Barrier. *Journal of the ACM (JACM)* 45, 5 (1998), 783–797.
[14] Andrew V. Goldberg and Robert Endre Tarjan. 1988. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)* 35, 4 (1988), 921–940.
[15] Steve Hanneke. 2014. Theory of Disagreement-Based Active Learning. *Foundations and Trends in Machine Learning* 7, 2-3 (2014), 131–309.
[16] John E. Hopcroft and Richard M. Karp. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal of Computing* 2, 4 (1973), 225–231.
[17] Lars Kolb, Andreas Thor, and Erhard Rahm. 2012. Load Balancing for MapReduce-based Entity Resolution. In *Proceedings of International Conference on Data Engineering (ICDE)*. 618–629.
[18] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 1197–1208.
[19] Hanna Köpcke and Erhard Rahm. 2010. Frameworks for entity matching: A comparison. *Data Knowledge Engineering (DKE)* 69, 2 (2010), 197–210.
[20] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment (PVLDB)* 3, 1 (2010), 484–493.
[21] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. In *Proceedings of ACM Management of Data (SIGMOD)*. 1133–1147.
[22] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of machine learning.* MIT Press.
[23] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms.* Cambridge University Press.
[24] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*. 269–278.
[25] Yufei Tao. 2018. Entity Matching with Active Monotone Classification. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. ACM, 49–62.
[26] Andreas Thor and Erhard Rahm. 2007. MOMA - A Mapping-based Object Matching System. In *Proceedings of Biennial Conference on Innovative Data Systems Research (CIDR)*. 247–258.

A PROOF OF LEMMA 5

We will use the following standard form of Chernoff bounds [23]:

- for any $\gamma \in (0, 1]$:

$$\mathbf{Pr}\left[\left|\mu - \frac{1}{t}\sum_{i=1}^{t} X_i\right| \geq \gamma\mu\right] \leq 2\exp\left(-\frac{\gamma^2 t\mu}{3}\right); \quad (35)$$

- for any $\gamma \geq 0$:

$$\mathbf{Pr}\left[\frac{1}{t}\sum_{i=1}^{t} X_i \geq (1+\gamma)\mu\right] \leq \exp\left(-\frac{\gamma^2}{2+\gamma}t\mu\right). \quad (36)$$

Case 1: $\mu \geq \phi$.

$$\begin{aligned}\mathbf{Pr}\left[\left|\mu - \frac{1}{t}\sum_{i=1}^{t} X_i\right| \geq \phi\right] &= \mathbf{Pr}\left[\left|\mu - \frac{1}{t}\sum_{i=1}^{t} X_i\right| \geq \frac{\phi}{\mu}\cdot\mu\right] \\ &\overset{\text{(by (35))}}{\leq} 2\exp\left(-\frac{1}{3}\left(\frac{\phi}{\mu}\right)^2 t\mu\right)\end{aligned}$$

which is at most δ when $t = \lceil \frac{3\mu}{\phi^2}\ln\frac{2}{\delta}\rceil$.

Case 2: $\mu < \phi$.

$$\begin{aligned}&\mathbf{Pr}\left[\left|\mu - \frac{1}{t}\sum_{i=1}^{t} X_i\right| \geq \phi\right] \\ =\ &\mathbf{Pr}\left[\frac{1}{t}\sum_{i=1}^{t} X_i \geq \phi + \mu\right] = \mathbf{Pr}\left[\frac{1}{t}\sum_{i=1}^{t} X_i \geq \left(1+\frac{\phi}{\mu}\right)\mu\right] \\ \overset{\text{(by (36))}}{\leq}\ &\exp\left(-\frac{(\phi/\mu)^2}{2+\phi/\mu}\cdot t\mu\right) = \exp\left(\frac{t\phi^2}{2\mu+\phi}\right)\end{aligned}$$

which is at most δ when $t = \lceil \frac{2\mu+\phi}{\phi^2}\ln\frac{1}{\delta}\rceil \leq \lceil \frac{3}{\phi}\ln\frac{1}{\delta}\rceil$.

B PROOF OF LEMMA 6

Construct an acyclic directed graph $G = (V, E)$ where $V = P$ (every point in P is a vertex) and $E = \{(u, v) \in P \times P \mid u \neq v \text{ and } u \succcurlyeq v\}$. The construction can be easily completed in $O(dn^2)$ time. Finding a chain decomposition of P with the least number (i.e., w) of chains is equivalent to computing the smallest number of vertex-disjoint paths to cover V. The latter problem can be converted[4] to computing a maximum matching of a bipartite graph with $O(|V|)$ vertices and $O(|E|)$ edges, which in turn can be solved by the Hopcroft-Karp algorithm [16] in $O(\sqrt{|V|}\cdot|E|)$ time. The lemma now follows from the fact that $|V| = n$ and $|E| = O(n^2)$.

[4] See https://en.wikipedia.org/wiki/Maximum_flow_problem.

C PROOF OF LEMMA 8

Let x be the minimum capacity of all source-sink cuts and y be the minimum weight of all cut-edge sets. We will prove $x = y$.

Proof of $x \geq y$. Let $(V_{\sqsubset}, V_{\sqsupset})$ be a source-sink cut with capacity x. Construct $E_{cut} = E \cap (V_{\sqsubset} \times V_{\sqsupset})$. E_{cut} is a cut-edge set because removing E_{cut} disconnects all paths from $V_{\sqsubset}$ to $V_{\sqsupset}$. Since the weight of E_{cut} is x, it follows that $y \leq x$.

Proof of $x \leq y$. Let E_{cut} be a cut-edge set with weight y. Construct $V_{\sqsubset}$ to include all vertices in V reachable from the source vertex, after removing the edges in E_{cut}; then, set $V_{\sqsupset} = V \setminus V_{\sqsubset}$. E_{cut} must include every edge $(u, v) \in E$ such that $u \in V_{\sqsubset}$ and $v \in V_{\sqsupset}$. Otherwise, the source vertex must be able to reach v even with E_{cut} removed, contradicting the fact that $v \notin V_{\sqsubset}$. It thus follows that the capacity of $(V_{\sqsubset}, V_{\sqsupset})$ is at most y, implying $x \leq y$.

D PROOF OF COROLLARY 20

We can treat A_{ran} as a random variable drawn from a set $\mathcal{A}$ of deterministic algorithms. We call an algorithm $A_{det} \in \mathcal{A}$ *accurate* if $nonoptcnt(A_{det}) \leq (2/5)n$. Define $\mathcal{A}_{acc}$ as the set of accurate algorithms in $\mathcal{A}$. Observe that $\mathbf{Pr}[A_{ran} \in \mathcal{A}_{acc}]$ must be at least $1/6$; otherwise, with probability at least $5/6$, $nonoptcnt(A_{ran}) > \frac{5}{6} \cdot \frac{2}{5} \cdot n = n/3$, giving a contradiction. By Lemma 19, however, every accurate $\mathcal{A}_{det}$ must satisfy $totalcost(A_{det}) = \Omega(n^2)$. Therefore, $\mathbf{E}[totalcost(A_{ran})] \geq \Omega(n^2) \cdot \mathbf{Pr}[A_{ran} \in \mathcal{A}_{acc}] = \Omega(n^2)$.