Adaptive Stratified Sampling for Precision-Recall Estimation

Ashish Sabharwal
Allen Institute for Artificial Intelligence
ashishs@allenai.org

Yexiang Xue
Purdue University
yexiang@cs.purdue.edu

Abstract

We propose a new algorithm for computing a constant-factor approximation of precision-recall (PR) curves for massive noisy datasets produced by generative models. Assessing validity of items in such datasets requires human annotation, which is costly and must be minimized. Our algorithm, ADASTRAT, is the first data-aware method for this task. It chooses the next point to query on the PR curve adaptively, based on previous observations. It then selects specific items to annotate using stratified sampling. Under a mild monotonicity assumption, ADASTRAT outputs a guaranteed approximation of the underlying precision function, while using a number of annotations that scales very slowly with N, the dataset size. For example, when the minimum precision is bounded by a constant, it issues only $\log \log N$ precision queries. In general, it has a regret of no more than $\log \log N$ w.r.t. an oracle that issues queries at data-dependent (unknown) optimal points. On a scaled-up NLP dataset of 3.5M items, ADASTRAT achieves a remarkably close approximation of the true precision function using only 18 precision queries, 13x fewer than best previous approaches.

1 INTRODUCTION

Generative machine learning models can produce massive amounts of noisy data. To be fruitfully used as a standalone resource for human consumption or in downstream applications, a practitioner must understand the quality of such data. This is often done with a precision-recall or PR curve, which characterizes how data quality degrades as the model's confidence in the validity of each item reduces. While a PR curve can be easily created for discriminative models by using pre-annotated held-out data, doing so for generative models is not straightforward. The latter is particularly challenging when human judgment or an expensive simulation is required to assess the validity or quality of generated data items.

Consider, for example, a creative deep learning system that can generate a million poems about a given topic (Ghazvininejad et al., 2016) or a natural language system that has produced over a hundred million English paraphrase pairs (Ganitkevitch et al., 2013; Pavlick et al., 2015). How does one go about assessing the quality of such generated data or of the models behind them?

A key bottleneck is annotation: Despite substantial advances in crowdsourcing technology, our ability to annotate novel data at a reasonable cost is far outpaced by increasingly sophisticated models that generate data at an even quicker pace. Computing the exact precision of a dataset of N items requires annotating the validity of every item, making exact computation infeasible for all but the smallest datasets. Conventional random sampling methods can achieve a constant-factor approximation of the PR curve with $\Theta(\sqrt{N \log N})$ valid/invalid annotations, but this, as Sabharwal and Sedghi (2017) argued, is also impractical in the modern era of big data. They proposed a logarithmic stratified sampling algorithm, henceforth referred to as LOGSTRAT, that can do so using only $\mathrm{O}(\log N \log \log N)$ annotations,[1] as long as the underlying precision function satisfies a *weak monotonicity* property. They also proposed PAULA, which achieves this with $\mathrm{O}(\Delta \log N)$ annotations, but requires a stronger notion of local monotonicity akin to concavity. This stronger monotonicity is characterized by a parameter Δ, which is difficult to estimate from data.

Both of these algorithms *query* the precision function at a set S of geometrically spaced points (thus $|S| = \log N$),

[1] These logarithms are w.r.t. base $1 + \epsilon$, the guaranteed approximation factor. The bounds thus scale roughly as $1/\epsilon$.

and interpolate between them. They, however, suffer from a limitation that S is chosen in a *data oblivious* way—it depends only on N and the desired approximation ratio, independent of the actual data. While $\log N$ queries are sufficient, they might be overkill, e.g., in the extreme case when the precision function is a constant.

We present a new algorithm, called ADASTRAT for adaptive stratifiled sampling, that *adaptively chooses what to query next based on current observations of the data*. It provides a guaranteed approximation under the same weak monotonicity condition as LOGSTRAT, without the stronger condition needed by PAULA.

The main novelty is the following: Given any k points observed on a PR curve, we show how to precisely characterize the "envelope" (Figure 1) of all possible PR curves that pass through these k points (Theorem 2). This envelope can be maintained efficiently as more points are observed. This leads to a natural bisection-style algorithm, which iterates until the "height" of the envelope (i.e., the maximum gap between its upper and lower boundaries) falls within the desired approximation ratio. The approximate curve ADASTRAT outputs is the geometric mean of the resulting upper and lower envelopes, which are *non-linear*, in line with the fact that a linear interpolation isn't appropriate in the precision-recall space (Davis and Goadrich, 2006).

ADASTRAT is surprisingly powerful both in theory and in practice. Formally, besides the initial few data points that each of these algorithm annotates, ADASTRAT uses $\mathrm{O}(K \log K)$ annotations chosen via adaptive stratified sampling (Theorem 8) if it ends up querying K points before meeting the stopping condition. The data determines how large K is. When the precision function decays very rapidly or very slowly, K can be as small as 2. Indeed, in two extreme cases, ADASTRAT queries only the first and last points of the PR curve and accurately interpolates everything in-between. When the minimum precision is bounded by a constant (e.g., 0.5 or 0.3) as in most practical cases, K scales as $\log \log N$ (Corollary 1). In the worst case, K is $\log N$ (Theorem 4), matching the asymptotic bound for LOGSTRAT.

We perform a *regret analysis* of ADASTRAT, showing (Theorem 6) that it never needs more than roughly $\log \log N$ times more queries than an "optimal" oracle algorithm that may use *a priori* knowledge of the shape of the precision function to decide which points to query.

Using the envelope view, we also provide a *matching lower bound*: every algorithm that operates by querying the precision function at some subset of points and guarantees a constant-factor approximation, must query $\Omega(\log N)$ points in the worst case (Theorem 7).

From a practical perspective, we evaluate various algorithms on scaled-up versions of the fully-annotated PPDB dataset used by Sabharwal and Sedghi (2017). On the PPDB-36K dataset with 35,615 items, we find that ADASTRAT *queries only 18 points* of the precision function, a 4.3x reduction from the 78 points queried by both LOGSTRAT and PAULA. Its strength is further highlighted by larger datasets, such as PPDB-100x, a 100x larger fully-annotated randomized variant that we created with a similar PR curve as the original. Here, despite the 100-fold increase in dataset size, ADASTRAT continues to query only 18 points, 13x fewer than the 234 needed by LOGSTRAT and PAULA. ADASTRAT uses mere 24K annotations,[2] a tiny fraction of the 3.5M items in this expanded dataset, while still yielding an impressive practical approximation (Figure 4).

1.1 RELATED WORK

Despite the importance of evaluating the precision-recall tradeoff of generative machine learning models, much research has been devoted to computing summary statistics (average precision AP, discounted cumulative gain DCG, etc.). Various results provide confidence intervals around estimated statistics (Carterette et al., 2006; Yilmaz et al., 2008; Aslam et al., 2006; Yilmaz and Aslam, 2006; Schnabel et al., 2016), often using different sampling approaches equipped with variance reduction techniques. Kanoulas (2015) provides a survey of relevant quality evaluation approaches in information retrieval.

In contrast to these efforts, we focus on characterizing the full precision recall curve at scale (over millions of items) and with provable guarantees. This task is considerably more challenging than computing summary statistics, an evidence of which is that these statistics can often be easily "read off" if one has computed the entire curve.

Relatively little research effort has been devoted to capturing an entire precision curve. In the area of vision, Welinder et al. (2013) propose semi-supervised performance evaluation, which is a generative model to capture a classifier's confidence scores. Unlike their use of a parametric model that makes certain assumptions about the curve, ours is a model-free approach relying only on a (weak form of) monotonicity.

Our setup is closest to that of Sabharwal and Sedghi (2017). Different from their approach, we propose to access the precision-recall curve in a data-aware, *adaptive* fashion. This, as we show, greatly reduces the sample complexity. Further, we do not make the strong monotonicity assumption needed for their strongest algorithm.

[2]The conventional method needs 284K annotations and LOGSTRAT needs 54K.

2 PRELIMINARIES

Consider the ranked output $T = (t_1, t_2, \ldots, t_N)$ of an algorithm $\mathcal{A}$, where each t_i comes from some universe U (e.g., all documents on the Web, all paraphrase pairs, all subject-verb-object triples, etc.). Each item $u \in U$ is associated with an unknown true label $v(u) \in \{0, 1\}$ that captures the semantics of some underlying task (e.g., whether a document is relevant to a query, whether a pair of phrases is a linguistic paraphrase, whether a triple denotes a true fact, etc.). We assume access to a noisy estimator, e.g., a crowd-sourced annotation, $\tilde{v}(u)$ of $v(u)$ that equals $1 - v(u)$ with probability $\eta < 1/2$, and equals $v(u)$ otherwise. The **precision function of $\mathcal{A}$**, $p : [N] \to [0, 1]$, maps each rank $r \in [N]$ to the fraction of the top r items in T that are positive, i.e., labeled as 1:

$$p(r) = \frac{1}{r} \sum_{i=1}^{r} v(t_i) \tag{1}$$

where we omit $\mathcal{A}$ from the notation for brevity.

Precision functions are widely used in machine learning. In fact, they are the building blocks of many statistical metrics. For example, a commonly used metric, precision-at-k, which measures the quality of the top-k ranked items, is exactly $p(k)$. As a second example, precision-recall curves can be built from precision functions. To see this, suppose a classifier outputs and ranks items based on its belief that each item is positive. $v(t_i)$ is an indicator variable, that is 1 if and only if the item ranked at the i-th place is positive. The classifier draws a line and classifies the top k items as positive examples. The precision of such a decision is $1/k \sum_{i=1}^{k} v(t_i)$, which is exactly $p(k)$, while the recall is $\sum_{i=1}^{k} v(t_i) / \sum_{i=1}^{N} v(t_i)$, which is $p(k)/p(N)$. Other metrics, such as Gain@k, accuracy, F1, true positive rate (TPR), false positive rate (FPR), Receiver Operating Characteristic (ROC) curve, average precision (AP), specificity, sensitivity, etc, can all be computed from p. Surveys by Fawcett (2006), Davis and Goadrich (2006), and Majnik and Bosnic (2013) provide more examples.

Given T, indirect access to $\tilde{v}$, and $\epsilon \in (0, 1]$, our goal is to compute a pointwise $(1 + \epsilon)$-approximation $\tilde{p}$ of p. We assume accessing each $\tilde{v}(t_i)$ is costly, e.g., needs human annotation. Therefore, we would like to compute $\tilde{p}$ efficiently in terms of the number of evaluations of $\tilde{v}$.

2.1 POINT ESTIMATES: RANDOM SAMPLING

A simple way to obtain an estimate $\tilde{p}(r)$ of $p(r)$ for a *fixed* rank r, which we refer to as a *point estimate* at r, is via random sampling: Sample (with repetition) a set of indices J independently and uniformly from $\{1, 2, \ldots, r\}$, obtain a noisy estimate $\tilde{v}(t_j)$ for each $v(t_j)$, and compute the empirical average $\tilde{p}(r) = \frac{1}{z} \sum_{j \in J} \tilde{v}(t_j)$ where $z = |J|$. Then, assuming $p \geq 1/3$, the expected value of $\tilde{p}(r)$ is within a factor of $1 + \eta$ of $p(r)$ (see Appendix). One can apply tail inequalities such as the two-sided Hoeffding bound (Hoeffding, 1963) to compute how tight the estimate is. For any $\epsilon > \eta$, to obtain a $(1+\epsilon)$-approximation of $p(r)$ with a confidence of $1 - \delta$ (e.g., a 95% confidence would mean $\delta = 0.05$), it suffices to have z samples where:

$$z \geq \frac{(1 + \eta)^2}{2(\epsilon - \eta)^2 p(r)^2} \ln \frac{2}{\delta}. \tag{2}$$

Details are deferred to the Appendix. When $\eta = 0$, this simplifies to the bound of Sabharwal and Sedghi (2017).

2.2 WEAK MONOTONICITY

Being the average of r 0-1 numbers, $p(r)$ necessarily fluctuates up and down as r increases. Nevertheless, we assume that $T = (t_1, t_2, \ldots, t_N)$ is a ranked output of an algorithm $\mathcal{A}$, where the true $v(t_i)$ in the beginning are more likely to be 1. In other words, one expects $p(r)$ to broadly decrease with increasing r. This property is captured by the following weak monotonicity notion introduced by Sabharwal and Sedghi (2017), for which we use a slightly different notation:

Definition 1 (Weak Monotonicity)**.** *Let $m, \tilde{r} \in \mathbb{N}^+$. Then p is* $(\tilde{r}, m)$-weak monotone *if for all $r_1 \geq \tilde{r}$ and $r_2 \geq r_1 + m$, we have $p(r_1) \geq p(r_2)$.*

Weak monotonicity guarantees that, after the first $\tilde{r}$ points, precision is non-increasing for points ranked at least m apart. Under this property, Sabharwal and Sedghi (2017) showed that it is sufficient to compute precision at only logarithmically many points in order to guarantee a tight approximation of the entire PR curve, which is reflect by their algorithm LOGSTRAT. They also relied on a stronger monotonicity assumption for their strongest algorithm, which we do *not* assume here.

Theorem 1 (LOGSTRAT (Sabharwal and Sedghi, 2017))**.** *Let $T, v, p, \tilde{r}, m$ be as above. Let $\epsilon \in (0, 1], \delta > 0, p_{min}$ be the minimum value of p, and $\beta > 1$. Let $\ell = \lceil \log_{1+\epsilon} \tilde{r} \rceil$ and $L = \lfloor \log_{1+\epsilon} N \rfloor$. If $m \leq \lfloor \epsilon(1 + \epsilon)^{\ell} - 1 \rfloor$ and p is $(\tilde{r}, m)$-monotone, then with probability at least $1 - \delta$, the output of* LOGSTRAT *on input $(T, v, \epsilon, \tilde{r}, \delta, p_{min}, \beta)$ is a $\beta(1 + \epsilon)$-approximation of $p(r)$. Further,* LOGSTRAT *queries p at $L - \ell$ points, and uses annotation of the first (roughly) $\tilde{r}$ points and of $\frac{\epsilon(L-\ell)}{2(\beta-1)^2(1+\epsilon)p_{min}^2} \ln \frac{L-\ell}{\delta/2}$ points chosen randomly via stratified sampling.*

Note that this result assumes the noiseless setting, $\eta = 0$. Note also that since $L = \Theta(\log N)$, LOGSTRAT re-

quires querying p at $\Theta(\log N)$ points and annotating $\Theta(\log N \log\log N)$ data points. Our goal is to improve upon this by adaptively deciding where to query (and which points to annotate), and when to stop.

3 CHARACTERIZING PRECISION FUNCTIONS THROUGH k POINTS

What could a precision function possibly look like if we know values of it at k points? We answer this question by providing a precise characterization of all precision functions passing through k given points, under the assumption of weak monotonicity. First, we characterize a tight upper bound $ub(v; y, p(y))$ and a lower bound $lb(v; y, p(y))$ for every point $p(v)$ at the precision function if we know the value of a single point $p(y)$. We call the space between ub and lb an *envelope* induced by the value of $p(y)$, because any $p(v)$ must be sandwiched between $lb(v; y, p(y))$ and $ub(v; y, p(y))$. These bounds are formally defined next, and illustrated in Figure 1.

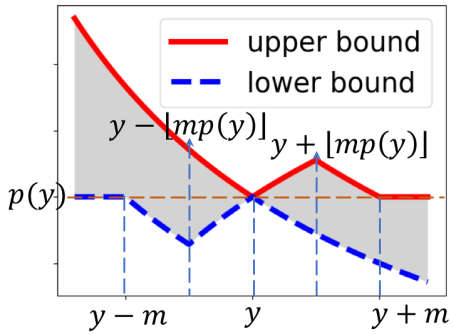

Figure 1: A graphical illustration of the upper bound ub (red line) and the lower bound lb (dashed blue line) induced by one point $p(y)$. The envelope is shaded.

Definition 2. *Let p be a precision function whose value $p(y)$ is known at a point y. Define ub and lb, each parameterized by $y, p(y)$, and (implicitly) by m, as:*

$$ub(v; y, p(y)) = \begin{cases} p(y)y/v & \text{if } v \le y \\ (p(y)y + v - y)/v & \text{if } y < v \le y + \lfloor mp(y) \rfloor \\ (p(y)y + \lfloor mp(y) \rfloor)/v & \text{if } y + \lfloor mp(y) \rfloor < v \le y + m \\ p(y) & \text{if } v > y + m \end{cases}$$

$$lb(v; y, p(y)) = \begin{cases} p(y), & \text{if } v < y - m, \\ (p(y)y - \lfloor mp(y) \rfloor)/v, & \text{if } y - m \le v < y - \lfloor mp(y) \rfloor \\ (p(y)y + v - y)/v, & \text{if } y - \lfloor mp(y) \rfloor \le v < y, \\ p(y)y/v, & \text{if } v \ge y. \end{cases}$$

Our characterization is summarized by the following theorem, whose proof is left to the appendix:

Theorem 2. *Let p be any $(\tilde{r}, m)$-monotonic precision function. Then, for any $v, y > \tilde{r}$, we have:*

$$lb(v; y, p(y)) \le p(v) \le ub(v; y, p(y))$$

Further, ub and lb are tight—each corresponds to a valid precision function whose value at y is $p(y)$.

This single point envelope characterization easily extends to the case where the values of p at are known at k points, $p(y_1), p(y_2), \ldots, p(y_k)$. The envelope here is the intersection of the k single point envelopes:

$$ub(v) = \min_{j=1}^{k} ub(v; y_j, p(y_j)) \tag{3}$$

$$lb(v) = \max_{j=1}^{k} lb(v; y_1, p(y_1)) \tag{4}$$

Finally, we define the *height* of the envelope induced by ub and lb as the maximum over i of $ub(i)/lb(i)$.

4 The AdaStrat ALGORITHM

Armed with the notion of an envelope characterizing all precision functions that could possibly pass through k observed points, we describe AdaStrat (Algorithm 1). The idea is to *query* p near the beginning and the end, compute the envelope induced by these two observations, and continue making further queries in the middle and tightening the envelope until its height is within (the square of) the desired approximation ratio. The algorithm then outputs the geometric mean of the (nonlinear) upper and lower bounds of the final envelope.

As before, $T = (t_1, t_2, \ldots, t_N)$ are the ranked data items with (unknown) true binary labels $v(t_i)$ and precision function p. We assume access to an η-noisy estimator $\tilde{v}$ of v and an oracle Query$(i, T, \tilde{v})$ that returns a guaranteed β-approximation of the true precision $p(i)$ at a given point i, for some $\beta \ge 1 + \eta$. Given $\epsilon, \delta > 0$, our goal is to obtain a $\beta(1+\epsilon)$-approximation of the entire p with confidence at least $1 - \delta$. For $m, \tilde{r} \in \mathbb{N}^+$, we assume p is $(\tilde{r}, m)$-weak monotonic. For brevity, we define:

$$\tilde{l} = \max\left\{ \left\lceil \frac{(1+\epsilon)^2 m}{2\epsilon + \epsilon^2} \right\rceil, \tilde{r} \right\}.$$

We first discuss a simple case, where Query$(i, T, \tilde{v})$ returns the exact value of $p(i)$, i.e., $\beta = 1$ (and thus $\eta = 0$). We will extend our result to the case where $\beta > 1$ later. In Algorithm (1), we maintain the envelope of possible precision functions represented by the upper bound $\tilde{ub}(i)$ and the lower bound $\tilde{lb}(i)$. We update these bounds in function UpdateUL as we get access to the values of the precision function at different locations. UpdateUL

Algorithm 1: ADASTRAT$(T, \tilde{l}, \tilde{v}, \epsilon)$: Adaptive Stratified Sampling for Approximating the Precision Function.

for $i = 1, 2, \ldots, N$ **do** $\tilde{ub}(i) \leftarrow 1$; $\tilde{lb}(i) \leftarrow 0$
for $i = 1, 2, \ldots, \tilde{l}$ **do**
  $v(t_i) \leftarrow \text{ACCESS}(i, T)$
  $\tilde{p}(i) \leftarrow \frac{1}{i}\sum_{j=1}^{i} v(t_j)$
  $\tilde{ub}, \tilde{lb} \leftarrow \text{UPDATEUL}(i, \tilde{p}(i), \tilde{ub}, \tilde{lb})$
$\tilde{p}(N) \leftarrow \text{QUERY}(N, T, \tilde{v})$
$\tilde{ub}, \tilde{lb} \leftarrow \text{UPDATEUL}(N, \tilde{p}(N), \tilde{ub}, \tilde{lb})$
$\tilde{p}(\tilde{l}+1), \ldots, \tilde{p}(N-1) \leftarrow \text{PR}(\tilde{l}, N, \tilde{p}(\tilde{l}), \tilde{p}(N), \tilde{ub}, \tilde{lb})$
return $\tilde{p}(1), \ldots, \tilde{p}(N)$

Function $\text{PR}(l, r, \tilde{p}(l), \tilde{p}(r), \tilde{ub}, \tilde{lb})$
  if $\max_{i \in \{l,\ldots,r\}} \frac{\tilde{ub}(i)}{\tilde{lb}(i)} \le (1+\epsilon)^2$ *or* $\frac{r}{l} \le (1+\epsilon)^2$ **then**
    // stopping condition met
    for $i \in \{l+1, \ldots, r-1\}$ **do**
      $\tilde{p}(i) \leftarrow \sqrt{\tilde{ub}(i)\,\tilde{lb}(i)}$
  else
    $c \leftarrow \text{round}(\sqrt{lr})$ // bisect the interval
    $\tilde{p}(c) \leftarrow \text{QUERY}(c, T, \tilde{v})$ // query mid-point
    $\tilde{ub}, \tilde{lb} \leftarrow \text{UPDATEUL}(c, \tilde{p}(c), \tilde{ub}, \tilde{lb})$
    $\tilde{p}(l+1), \ldots, \tilde{p}(c-1) \leftarrow \text{PR}(l, c, \tilde{p}(l), \tilde{p}(c), \tilde{ub}, \tilde{lb})$
    $\tilde{p}(c+1), \ldots, \tilde{p}(r-1) \leftarrow \text{PR}(c, r, \tilde{p}(c), \tilde{p}(r), \tilde{ub}, \tilde{lb})$
  return $\tilde{p}(l+1), \ldots, \tilde{p}(r-1)$

Function $\text{UPDATEUL}(y, \tilde{p}(y), \tilde{ub}, \tilde{lb})$:
  for $i = 1, \ldots, N$ **do**
    $\tilde{ub}(i) \leftarrow \min\{\tilde{ub}(i), ub(\tilde{v}; y, \tilde{p}(y))\}$
    $\tilde{lb}(i) \leftarrow \max\{\tilde{lb}(i), lb(\tilde{v}; y, \tilde{p}(y))\}$
  return $\tilde{ub}, \tilde{lb}$

intersects the old envelope with a new pointwise upper and lower bound, just as in Equation (3,4). We compute the exact values of $p(1), \ldots, p(\tilde{l})$ by accessing the values of $v(t_1), \ldots, v(t_{\tilde{l}})$ directly. Here, $\text{ACCESS}(i, T)$ returns the exact value of $v(t_i)$.[3]

Function PR returns $\tilde{p}(l+1), \ldots, \tilde{p}(r-1)$, which form an $(1+\epsilon)$-approximation to the true values $p(l+1), \ldots, p(r-1)$. In function PR, first the algorithm checks the height of the envelope between $\tilde{p}(l)$ and $\tilde{p}(r)$. If the height is less than $(1+\epsilon)^2$, then the algorithm stops, returning the geometric mean of $\tilde{ub}$ and $\tilde{lb}$. When $\beta = 1$, the second stopping condition $r/l \le (1+\epsilon)^2$ is redundant, because for any l, r, such that $\tilde{l} \le l < r < (1+\epsilon)^2 l$, we must have $p(r) \ge p(l)l/r \ge p(l)/(1+\epsilon)^2$ and $p(l) \ge p(r)(r-m)/l \ge p(r)/(1+\epsilon)^2$, due to Theorem 2. In other words, if condition $r/l < (1+\epsilon)^2$ is met, then the height of the envelope has already dropped below $(1+\epsilon)^2$. If the function does not stop, there is at least one point i between l and r, where $\tilde{ub}(i)/\tilde{lb}(i)$ exceeds $(1+\epsilon)^2$. In this case, we query the function value at a middle point $c = \text{round}(\sqrt{lr})$, and recursively call PR on intervals $(\tilde{p}(l), \ldots, \tilde{p}(c))$ and $(\tilde{p}(c), \ldots, \tilde{p}(r))$.

When $\beta > 1$, we stop first when the estimated boundaries $\tilde{ub}$ and $\tilde{lb}$ are within $(1+\epsilon)^2$. In this case, we know that true values of p lie in the range between $\beta\tilde{ub}$ and $\tilde{lb}/\beta$, which are at most $\beta^2(1+\epsilon)^2$ apart. It is easy to see that $\tilde{p} = \sqrt{\tilde{ub}\,\tilde{lb}}$ provides a $\beta(1+\epsilon)$ approximation to any curve in this range, which includes p. We also stop when $r/l \le (1+\epsilon)^2$. In this case, we know that the actual height of the envelope (distance between the true boundaries ub and lb) is bounded by $(1+\epsilon)^2$ (due to the same reason as why $r/l \le (1+\epsilon)^2$ is redundant when $\beta = 1$). Since all point estimations are at most off by β, $\tilde{ub}$ is at most $\beta\, ub$ and $\tilde{lb}$ is at least lb/β. Therefore, $\tilde{p} = \sqrt{\tilde{ub}\,\tilde{lb}}$ is a $\beta(1+\epsilon)$ approximation. Putting this all together, we have the following theorem:

Theorem 3. *Let $T, \tilde{v}, p, m, \tilde{r}, \tilde{l}, \beta$, and ϵ be as defined before. If the precision function p is $(\tilde{r}, m)$-weak monotonic and $\text{QUERY}(i, T, \tilde{v})$ is a β-approximation of $p(i)$ for all i, then the output of ADASTRAT (Algorithm 1) on input $(T, \tilde{l}, \tilde{v}, \epsilon)$ is a pointwise $\beta(1+\epsilon)$-approximation of the true precision values $p(1), \ldots, p(N)$.*

Sufficient Conditions for Stopping

To understand the complexity of ADASTRAT in terms of the number of calls to QUERY, we analyze the stopping condition of PR, namely, whether the height of the envelope is within than $(1+\epsilon)^2$. We provide two sufficient conditions for stopping. The two lemmas below follow by writing down the pointwise envelopes induced by $\tilde{p}(l)$ and $\tilde{p}(r)$ and making use of the fact that $r > l \ge \tilde{l}$.

Lemma 1. *Under weak monotonicity, if $\tilde{p}(l)/\tilde{p}(r) \le (1+\epsilon)^2$, the height of the envelope[4] is bounded by $(1+\epsilon)^2$.*

Lemma 2. *Under weak monotonicity, if $(\tilde{p}(r)r)/(\tilde{p}(l)l) \le (1+\epsilon)^2$, the height of the envelope is bounded by $(1+\epsilon)^2$.*

The sufficient stopping conditions captured by Lemmas 1 and 2 are two interesting cases of early stopping, in

[3] For simplicity, we assume ACCESS uses v instead of $\tilde{v}$. Under the noisy setting where ACCESS uses $\tilde{v}$, the results can be extended by averaging multiple calls to ACCESS.

[4] defined by substituting $\tilde{p}$ into (3) and (4).

contrast to LOGSTRAT, where $O(\log_{1+\epsilon} N)$ queries are needed regardless of the shape of the precision function.

Lemma 1 captures the case where p does not drop too much from l to r. This corresponds to the density of $v(t_i)$ that are 1 staying almost the same for all entries in the range from l to r. Notice that the density almost always cannot increase, because of weak monotonicity.

Lemma 2 captures the other extreme, where $v(t_i)$ is almost always zero for the entries in the range from l to r. In this case, the precision function drops at its fastest rate. Our algorithm is able to capture these two cases, stopping early, thereby preventing unnecessary queries.

Upper Bound on the Number of Query Calls

The above stopping conditions imply that ADASTRAT never makes more calls to QUERY than LOGSTRAT does. Specifically, deferring a proof to the Appendix:

Theorem 4. *Under the conditions of Theorem 3, the number of calls to* QUERY *is at most* $\log_{1+\epsilon}(N/\tilde{l})$.

4.1 REGRET BOUNDS

Consider an "optimal" algorithm that is guaranteed to produce a $(1+\epsilon)$-approximation of all weak monotonic precision function with as few accesses to QUERY as possible. If this algorithm knew the shape of p *a priori*, it could clearly be very smart about where it queries p in order to generate a guaranteed approximation. The *regret* of any algorithm, then, is defined as how many (multiplicatively) more accesses to QUERY it needs, compared to this optimal algorithm who knows all. We prove that ADASTRAT has a regret of no more than $\log_2 \log_{1+\epsilon} N$.

We start by exploring how such an "optimal" algorithm might behave. Suppose it has access to the maximum and minimum precision values, $p_{\max}$ and $p_{\min}$, as well as to $q_1, \ldots, q_K$, where $K = \left\lceil \log_{1+\epsilon} \frac{p_{\max}}{p_{\min}} \right\rceil$ and q_i is the *first* location where p falls below $p_{\max}/(1+\epsilon)^{i-1}$. Then, as we show next, it suffice for the "optimal" algorithm to make only K queries, namely to $p(q_1), \ldots, p(q_K)$, to guarantee a $(1+\epsilon)$-approximation:

Lemma 3. *Let $q_1, \ldots, q_K$ be as defined above. Let $\tilde{p}(j) = p(q_i)$ whenever $j \in \{q_i, \ldots, q_{i+1}\}$. Then $\tilde{p}$ is a $(1+\epsilon)$-approximation of p in the range $[l', N]$.*

Lemma 3 guarantees that the "optimal" algorithm does not make too many queries when the precision function decays slowly, i.e., $p_{\max}/p_{\min}$ is small. In the other extreme, where the precision function decays in its fastest possible way, $p(r)r$ stays almost as a constant. In this case, we can prove that the "optimal" algorithm does not make much more queries beyond the ratio of the maximal and minimal values of $p(r)r$. Specifically, suppose the "optimal" algorithm has access to $s_1, \ldots, s_P$, where s_j is the first location that function $p(r)r$ goes above $p(\tilde{l})\tilde{l}(1+\epsilon)^{j-1}$. Then $P = \left\lceil \log_{1+\epsilon} \frac{p(N)N}{p(\tilde{l})\tilde{l}} \right\rceil$. We can also prove that it suffices for the "optimal" algorithm to query the above P points to obtain a $(1+\epsilon)$-approximation:

Lemma 4. *Let $s_1, \ldots, s_P$ be as defined above. Let $\tilde{p}(j) = p(s_i)s_i/j$ whenever $j \in \{s_i, \ldots, s_{i+1}\}$. Then $\tilde{p}$ is a $(1+\epsilon)$-approximation of p in the range $[\tilde{l}, N]$.*

Proofs of these two lemmas may be found in Appendix B. Putting these together gives a bound on OPT, the number of times the optimal algorithm calls QUERY:

Theorem 5. *Under the conditions of Theorem 3,*

$$\mathrm{OPT} \le \left\lceil \log_{1+\epsilon} \min\left(\frac{p_{\max}}{p_{\min}}, \frac{p(N)N}{p(\tilde{l})\tilde{l}} \right) \right\rceil.$$

Now we state our main regret bound:

Theorem 6. *Under the conditions of Theorem 3,* ADASTRAT *calls* QUERY *no more than* $(\mathrm{OPT}+1)(1+\log_2 \log_{1+\epsilon} N)+1$ *times.*

This says that the number of QUERY calls made by ADASTRAT is roughly $O(\mathrm{OPT} \cdot \log_2 \log_{1+\epsilon} N)$. The high level idea to prove Theorem 6 is as follows. Suppose $r_1, \ldots, r_{\mathrm{OPT}}$ are the actual query points of the optimal algorithm. Because ADASTRAT uses a binary search, i.e., it always splits an interval at its geometric middle point. Then it takes ADASTRAT roughly $O(\log_2 \log_{1+\epsilon} N)$ splits to "locate" one query point r_i of the optimal algorithm (more precisely, find a point that is sufficiently close to r_i that guarantees the approximation bound). Hence, the total number of queries of ADASTRAT is bounded by OPT times $\log_2 \log_{1+\epsilon} N$. Our actual proof to Theorem 6 is based on walking through the actual calling map of the function PR, where each node in this map represents an actual interval $(p(l), p(r))$ that PR called. We leave this proof to Appendix B.

Combining Theorems 5 and 6, we immediately obtain the following worst case upper bound for ADASTRAT.

Corollary 1. *Under the conditions of Theorem 3,* ADASTRAT *calls* QUERY *no more than*

$$O\left(\log_{1+\epsilon} \min\left\{ \frac{p_{\max}}{p_{\min}}, \frac{p(N)N}{p(\tilde{l})\tilde{l}} \right\} \cdot \log_2 \log_{1+\epsilon} N \right)$$

times.

Thus, ADASTRAT makes very few queries when p is flat or decays very fast. In general, when $p_{\min}$ may be treated as a constant bounded away from zero (e.g., 0.5 or 0.3, as is the case in many practical applications), this shows that ADASTRAT scales essentially as $\log \log N$.

4.2 ASYMPTOTIC LOWER BOUND

What is the minimum number of calls to the QUERY *function needed in order to guarantee an* $(1+\epsilon)$*-approximation to* p*?* We provide a worst-case lower bound, confirming that ADASTRAT is asymptotically optimal in terms of the number of queries.

Theorem 7. *Let* $\mathcal{A}$ *be any algorithm that accesses the precision function only via the* QUERY *oracle and, for any* $(\tilde{r}, m)$*-weak monotonic precision function, outputs a curve that* $(1+\epsilon)$*-approximates it. For any* $\epsilon' > \epsilon$*,* $\mathcal{A}$ *must make at least* $\Omega(\log_{1+\epsilon'} N)$ *accesses to* QUERY.

The high level idea of the proof to Theorem 7 is as follows: let $J \approx \log_{1+\epsilon'} N$. We carefully construct a family of 2^J valid precision functions $F = \{f_0, f_1, \ldots, f_{2^J-1}\}$ such that, for any two functions f_i and f_j, there exists at least one point $y_{i,j}$, such that $f_i(y_{i,j})$ and $f_j(y_{i,j})$ are separated by more than $(1+\epsilon)^2$ (i.e., either $f_i(y_{i,j}) > (1+\epsilon)^2 f_j(y_{i,j})$ or $f_j(y_{i,j}) > (1+\epsilon)^2 f_i(y_{i,j})$). We call this point $y_{i,j}$ a *separating point* between f_i and f_j.

Now suppose algorithm $\mathcal{A}$ can output a $(1+\epsilon)$-approximation to any given precision function. Starting with an unknown function $f \in F$, we can use $\mathcal{A}$ to identify f. To do so, we run $\mathcal{A}$ to obtain a $(1+\epsilon)$-approximate curve $\tilde{f}$ and examine its values at all separating points. Because $\tilde{f}$ is a $(1+\epsilon)$-approximation and the distance between two functions at a separating point is more than $(1+\epsilon)^2$, we can unambiguously determine the correct f. Appendix B includes a detailed construction of the function family F following this high-level idea.

4.3 STRATIFIED SAMPLING FOR QUERY

Suppose ADASTRAT ends up calling QUERY on the K points $r_1 < r_2 \ldots < r_K$ (generally not in this order) before terminating. By design, $r_1 > \tilde{l}$ and $r_K = N$. Let $\delta > 0$ and $\beta > 1 + \eta \geq 1$. We would like QUERY to provide a β-approximation of $p(r_i)$ for all $i \in \{1, \ldots, K\}$ with an overall (cumulative) confidence of at least $1-\delta$. To achieve this, QUERY proceeds similarly to LOGSTRAT but with $\tilde{v}$ rather than v: it uses as an estimate of $p(r_i)$ the empirical average of η-noisy estimates $\tilde{v}(t_j)$ of true labels $v(t_j)$ for s uniform random samples j drawn independently from $[1, r_i]$, where:

$$s = \left\lceil \frac{(1+\eta)^2}{2(\beta-1-\eta)^2 p_{\min}^2} \ln \frac{2K}{\delta} \right\rceil \tag{5}$$

Here $p_{\min}$ is an estimate of (a lower bound on) the minimum value of p for the given data.[5]

Of course, we don't know K *a priori*; we will address this shortly. It follows from the Hoeffding bound, Eq. (2), that such an empirical average provides a β-approximation of $p(r_i)$ with confidence at least $1-\delta/K$. Applying the union bound over all i, ADASTRAT has overall confidence at least $1-\delta$ in its estimates being correct simultaneously at all K points $r_1, \ldots, r_K$.

As in LOGSTRAT, since we rely only on the union bound, the samples obtained for r_1 can be (partially) reused as samples for all $r_i > r_1$. The amount of reuse is determined by what we will refer to as the *sample density* of an interval in $\{1, \ldots, N\}$, defined as the ratio of the number of samples in this interval to the size of the interval. Clearly, in order to have s uniform samples available for r_i, we must have a sample density of at least s/r_i in the interval $[1, r_i]$. We would like to achieve this while minimizing the total number of samples.

It can be verified that the following *stratified sampling strategy*, henceforth referred to as $\mathcal{S}$, results in the minimum overall number of samples while ensuring that the sample density in $[1, r_i]$ is at least s/r_i:

$$\begin{aligned} &\text{draw } \left\lceil \tfrac{(r_1-\tilde{l})s}{r_1} \right\rceil \text{ samples in } [\tilde{l}+1, r_1] \\ &\text{draw } \left\lceil \tfrac{(r_i-r_{i-1})s}{r_i} \right\rceil \text{ samples in } [r_{i-1}+1, r_i] \text{ for } i > 1 \end{aligned}$$

In LOGSTRAT, the K points are visited in increasing order, simplifying the implementation of $\mathcal{S}$ in practice. Further, K is known *a priori* to be $\log_{1+\epsilon} N/\tilde{l}$ and r_i by design equals $r_{i-1}(1+\epsilon)$. This makes it easy to compute the total number of evaluations of v needed, which sums up to $\tilde{l} + \frac{\epsilon}{1+\epsilon} s \log_{1+\epsilon} N/\tilde{l}$, in line with Theorem 1.

The adaptive nature of ADASTRAT makes both the implementation of $\mathcal{S}$ and a similar calculation challenging. Nevertheless, the following result holds:

Theorem 8. *Under the conditions of Theorem 3, for any* $\delta > 0$ *and* $\beta > 1 + \eta \geq 1$*,* QUERY *can be implemented using a stratified sampling strategy such that* ADASTRAT *provides a* $\beta(1+\epsilon)$*-approximation of the precision function* p *with a confidence of at least* $1-\delta$ *using* $\tilde{l}$ *evaluations of the true label* v *and:*

$$\left(\left\lceil \frac{r_1 - \tilde{l}}{r_1} \right\rceil + \sum_{i=2}^{K} \left\lceil \frac{r_i - r_{i-1}}{r_i} \right\rceil \right) \cdot s$$

evaluations of the noisy estimate $\tilde{v}$*, where* $s = \left\lceil \frac{(1+\eta)^2}{2(\beta-1-\eta)^2 p_{\min}^2} \ln \frac{2K}{\delta} \right\rceil$ *and* $r_1, r_2, \ldots, r_K$ *are the points where* ADASTRAT *calls* QUERY.

[5] Domain knowledge about the data might allow using a small constant, such as 0.3, for $p_{\min}$. Alternatively, one can use an estimate of $p(N)$ obtained via an *adaptive concentration inequality*, such as Corollary 1 of Zhao et al. (2016), which provides a dynamic stopping condition to decide how many samples are sufficient, and guarantees that this number, $z_{\min}$, is upper bounded by a generalization of Hoeffding's bound with an additional $\log\log$ term: $1.8 z_{\min}(\gamma^2 p(N)^2 - 0.6 \log(\log_{1.1} z_{\min} + 1)) \leq \ln(12/\delta)$, where $1 + \gamma = \frac{\beta}{1+\eta}$.

We note that this quantity is bounded above by Ks, which scales as $O(K \log K)$, considering other parameters as constants. This simplified expression is equivalent to *not* reusing samples at all, and thus quite loose in practice. Even so, for datasets requiring $K \ll \log N$, this is substantially smaller than the equivalent $O(\log N \log \log N)$ expression for LOGSTRAT.

Unlike LOGSTRAT, there are two hurdles to implementing a stratified sampling strategy that supports the number of annotations claimed in Theorem 8: K is unknown in the beginning and ADASTRAT does not visit $r_1, \ldots, r_K$ in increasing order. Let $r'_1, \ldots, r'_K$ be the order in which ADASTRAT actually queries the K points. To address the first hurdle (unknown K), we follow an **iterative deepening approach** and simply begin by assuming $K = 1$ when querying r'_1. When ADASTRAT decides to make the next query at r'_2, we set $K = 2$, go back to r'_1 to obtain correspondingly more annotations for it, and then obtain samples for r'_2 based on $K = 2$. This process continues, slowly incrementing K and obtaining more samples at previously queried points to make up for the difference. Since the samples are drawn independently, this yields the same outcome as if we had known the true value of K in advance and obtained the corresponding number of samples for each r'_i in a single shot.

To address the second hurdle (queries not in increasing order), we **adapt stratified sampling** as follows. For simplicity of exposition, we assume here that K is known at the start. When querying r'_1, we use stratification similar to LOGSTRAT and obtain $s(r'_1 - \tilde{l})/r'_1$ fresh samples in the range $[\tilde{l}+1, r'_1]$, inducing a sample density s/r'_1 in this range. When querying r'_2, there are two possibilities. If $r'_2 > r'_1$, then again we obtain fresh samples with density s/r'_2 in the range $[r'_1+1, r'_2]$, similar to LOGSTRAT. If, on the other hand, $r'_2 < r'_1$, we obtain fresh samples instead in the range $[\tilde{l}+1, r'_2]$ to increase the sample density here from s/r'_1 to s/r'_2. This process continues with each new query, whose effect is to raise the sample density between the immediately lower queried point and the current point. It can be verified that this process ends with the sample density underlying the expression in Theorem 8, namely s/r_i in the range $[r_{i-1}+1, r_i]$.

5 EXPERIMENTS

For an empirical evaluation, we consider the fully-annotated subset of PPDB 2.0 (Ganitkevitch et al., 2013) used by Sabharwal and Sedghi (2017), henceforth referred to as PPDB-36K. It contains $N = 35{,}615$ English language paraphrase pairs for each of which Pavlick et al. (2015) provide a correctness confidence score (thus inducing an overall ranking) obtained using a machine learning algorithm, as well as crowdsourced annotations

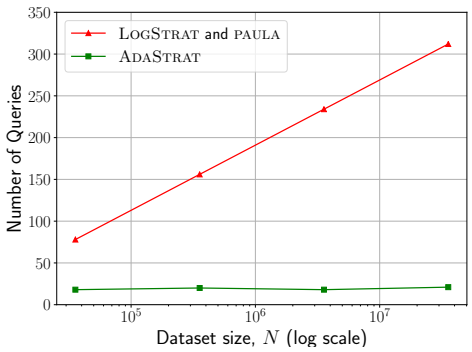

Figure 2: Number of queries of the precision function as dataset size increases. ADASTRAT uses only 21 queries even for PPDB-36K-1000x with 35M items.

of the validity of each pair as a valid paraphrase, on a 5-point scale (1-5). A pair t_i receiving an average human judgment of at least 3 is considered correct, i.e., $v(t_i) = 1$ for such i, and 0 otherwise. For a direct comparison with prior work, we experiment with noiseless access to v, i.e., $\eta = 0$ and $\tilde{v} = v$.

Given the fully-annotated nature of PPDB-36K, the true precision function for it can be easily calculated and used to assess the performance of algorithms such as ADASTRAT. One drawback of this dataset, however, is its relatively small size. To alleviate this while still retaining the property of having a fully-annotated yet realistic dataset, we consider *scaled up variants* of PPDB-36k, created as follows. Using a sliding window of size $\Delta = 100$, we compute the running average $q(i)$ of $v(t_i)$ for $1 \le i \le N$, using smaller sliding windows as appropriate when $i < \Delta$ or $i > N - \Delta$. For a scaling factor $s \in \{10, 100, 1000\}$, for each $1 \le i \le N$, we draw s independent random samples from the Bernoulli distribution with parameter $q(i)$. This results in 3 datasets, PPDB-36K-10x, PPDB-36K-100x, and PPDB-36K-1000x, that are 10, 100, and 1000 times larger than PPDB-36K, resp.

5.1 SCALING: QUERIES AND ANNOTATIONS

Our first experiment evaluates the number of queries (of the precision function) used by various algorithms, as well as the total number of annotations, as the dataset size is varied from 36K to 35M. We use the following parameters throughout: $\epsilon = 0.03, \delta = 0.05, \beta = 1.05$.[6]

Figure 2 shows in a semi-log plot the number of queries needed, as the dataset size grows.[7] As expected, both LOGSTRAT and PAULA use the same number of queries, which starts with 78 for PPDB-36K and grows propor-

[6] Code and data available at http://allenai.org.

[7] The exact number of queries is reported in Appendix A.

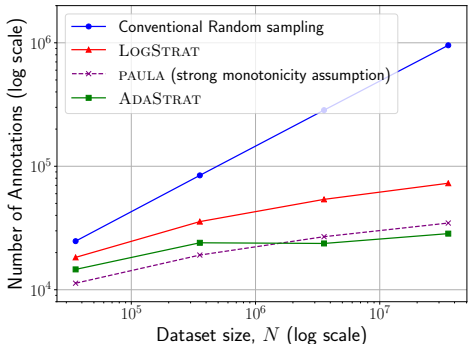

Figure 3: Number of annotations needed by various algorithms as dataset size increases. AdaStrat needs substantially fewer annotations than competing algorithms that also do not assume strong monotonicity.

tional to $\log N$, reaching 312 queries for PPDB-36K-1000x. In contrast, AdaStrat uses only 18 to 21 queries, even for the largest dataset with over 35M items. This aligns with the intuition that the adaptive nature of AdaStrat allows it to be driven more by the "shape" of the precision function, rather than by the raw data size.

Figure 3 illustrates in a log-log plot the total number of samples used by each method, as the dataset size grows; again, exact numbers may be found in Appendix A. The conventional random sampling baseline asymptotically scales as $\Theta(\sqrt{N \log N})$. In line with this, the corresponding blue curve has a slope of roughly 0.5, reaching close to 1M required annotations for PPDB-36K-1000x. LogStrat (red curve) is substantially more practical, growing from 18K annotations to 73K. paula (dashed purple line) needs the fewest annotations for the two smaller datasets, but relies the assumption of strong local monotonicity, which is difficult to verify in practice. Finally, AdaStrat (green line) uses the fewest number of annotations (24K and 28K, resp.) for the two larger datasets. Further, among algorithms that do not rely on strong monotonicity, AdaStrat has 20%-61% higher annotation efficiency than LogStrat and 41%-97% higher than conventional random sampling.

5.2 APPROXIMATION QUALITY

The top plot in Figure 4 shows the approximate precision function $\tilde{p}$ (green curve) produced by AdaStrat for PPDB-36K-100x. Despite querying only 18 points (marked with small red squares) along the true curve, AdaStrat is able to obtain a remarkably good approximation of the entire true precision function (shown in black, and often occluded by the green curve).

Both LogStrat and paula (bottom plot, red) also obtain a similarly tight approximation, except towards the

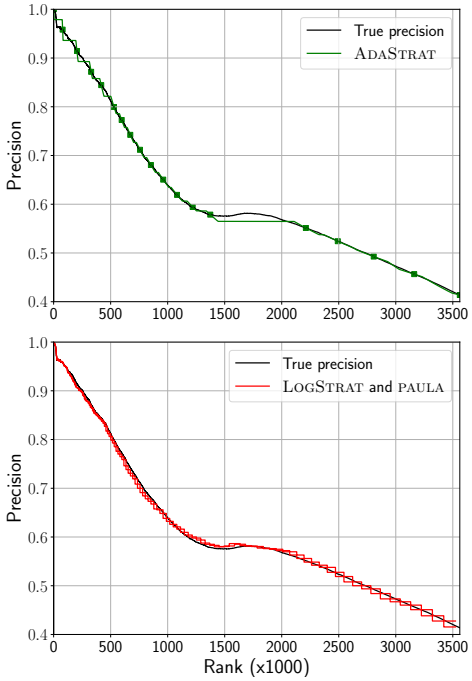

Figure 4: Precision function approximations generated by AdaStrat (top) and LogStrat and paula (bottom), for PPDB-36K-100x. 18 green markers (top) and 234 corners of red boxes (bottom) are the points queried by the corresponding algorithm.

right end of the curve. Importantly, however, they do so by querying the true precision at 234 points, visually identifiable as the "corners" of the little red boxes. In particular, because of the geometrically spaced nature of the points they query, there is an enormous number of queries in the left part of the curve, which, as illustrated by AdaStrat's more spaced-out query points, is unnecessary. This demonstrates the strength of AdaStrat in exploiting data observations to be smart about where and how often to query the true precision function.

6 CONCLUSION

We proposed AdaStrat, a data-aware algorithm for computing the precision function of massive noisy datasets, with a constant-factor approximation guarantee. AdaStrat intelligently chooses precision points to query. Under a mild monotonicity assumption, it outputs a guaranteed curve with minimal queries made to the PR curve, scaling very slowly with N, the number of items. AdaStrat's regret w.r.t. an oracle is bounded by $\log \log N$. We also provide a matching asymptotic lower bound in terms of the number of queries. On an NLP dataset of 3.5M items, AdaStrat achieves a close approximation with merely 18 precision queries.

References

J. A. Aslam, V. Pavlu, and E. Yilmaz. A statistical method for system evaluation using incomplete judgments. In *SIGIR*, 2006.

B. Carterette, J. Allan, and R. K. Sitaraman. Minimal test collections for retrieval evaluation. In *SIGIR*, 2006.

J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, 2006.

T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

J. Ganitkevitch, B. V. Durme, and C. Callison-Burch. PPDB: The paraphrase database. In *HLT-NAACL*, 2013.

M. Ghazvininejad, X. Shi, Y. Choi, and K. Knight. Generating topical poetry. In *EMNLP*, 2016.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

E. Kanoulas. A short survey on online and offline methods for search quality evaluation. In *RuSSIR*, 2015.

M. Majnik and Z. Bosnic. Roc analysis of classifiers in machine learning: A survey. *Intell. Data Anal.*, 17: 531–558, 2013.

E. Pavlick, P. Rastogi, J. Ganitkevitch, B. Van Durme, and C. Callison-Burch. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *ACL*, 2015.

A. Sabharwal and H. Sedghi. How good are my predictions? efficiently approximating precision-recall curves for massive datasets. In *UAI*, 2017.

T. Schnabel, A. Swaminathan, P. I. Frazier, and T. Joachims. Unbiased comparative evaluation of ranking functions. In *ICTIR*, 2016.

P. Welinder, M. Welling, and P. Perona. A lazy man's approach to benchmarking: Semisupervised classifier evaluation and recalibration. In *CVPR*, 2013.

E. Yilmaz and J. A. Aslam. Estimating average precision with incomplete and imperfect judgments. In *CIKM*, 2006.

E. Yilmaz, E. Kanoulas, and J. A. Aslam. A simple and efficient sampling method for estimating ap and ndcg. In *SIGIR*, 2008.

S. Zhao, E. Zhou, A. Sabharwal, and S. Ermon. Adaptive concentration inequalities for sequential decision problems. In *NIPS*, 2016.

A APPENDIX: EXPERIMENTS

Table 1 shows the number of queries for various algorithms and datasets, as used in Figure 2. Similarly, Table 2 shows the total number of annotations used, as depicted in Figure 3.

N	LOGSTRAT	PAULA	ADASTRAT
35615	78	78	18
356150	156	156	20
3561500	234	234	18
35615000	312	312	21

Table 1: Number of queries of the precision function issued by various algorithms for different size variants of PPDB-36K.

N	RANDOM	LOGSTRAT	PAULA	ADASTRAT
35615	24745	18287	11292	14612
356150	84369	35631	19092	24004
3561500	284834	53937	26892	23707
35615000	954359	72867	34692	28477

Table 2: Number of annotations used by various algorithms for different size variants of PPDB-36K.

B APPENDIX: PROOFS

B.1 Noisy Samples for Point Estimates

Consider the setting of Section 2.1, where we use noisy estimates $\tilde{v}(u)$ of $v(u)$ at J randomly chosen points from $\{1, 2, \ldots, r\}$, in order to estimate $p(r)$.

First, fix a u. Because of the error probability $\eta < 1/2$ in the estimate, the expected value $\tilde{v}(u)$ is $v(u)(1-\eta) + (1-v(u))\eta$, which equals $v(u) + \eta - 2\eta v(u)$. Second, note that the expected value of $\sum_{j\in J} v(t_j)$ across random choices of J is precisely $zp(r)$, where $z = |J|$.

Putting these together, we obtain that the expected value of $\frac{1}{z}\sum_{j\in J}\tilde{v}(t_j)$ across random choices of J is $p(r) + \eta - 2\eta p(r)$. Assuming $p(r) \le 1/3$, this lies within $(1 \pm \eta)\, p(r)$ and is also within a multiplicative factor of $1+\eta$ of $p(r)$.

Given $\epsilon > \eta$, we can now use the two-sided Hoeffding bound (Hoeffding, 1963) to assess how many samples, z, are needed to obtain a $(1+\epsilon)$-approximation of $p(r)$. For this, it suffices to use a z that guarantees a $\frac{1+\epsilon}{1+\eta}$-approximation of the expected value of $\frac{1}{z}\sum_{j\in J}\tilde{v}(t_j)$, which, as shown above, is always within a factor of $1+\eta$ of $p(r)$. To this end, a direct application of Hoeffding's inequality, while observing that $\frac{1+\epsilon}{1+\eta} - 1 = \frac{\epsilon-\eta}{1+\eta}$, yields:

$$z \ge \frac{(1+\eta)^2}{2(\epsilon-\eta)^2 p(r)^2} \ln\frac{2}{\delta}.$$

B.2 Proof of Theorem 2

The upper bound: when $v' \le y$, $p(v')v' \le p(y)y$ because of the monotonicity of the partial sum $\sum_{j=1}^{n} v(t_j)$. Therefore, $p(v') \le p(y)y/v'$, which is the result on the first line. When $v' > y + m$, $p(v') \le p(y)$ because of weak $(\tilde{r}, m)$-monotonicity, which is the result on the fourth line. In the middle, for $y + \lfloor mp(y) \rfloor < v' \le y + m$, because of weak $(\tilde{r}, m)$-monotonicity, $p(y+m) \le p(y)$, hence at most $p(y)$ fraction of the entries $v(t_{y+1}), v(t_{y+2}), \ldots, v(t_{y+m})$ can be 1. This implies that $p(v')v' \le p(y)y + \lfloor mp(y) \rfloor$, which is the third line. For $y < v' \le y + \lfloor mp(y) \rfloor$, in the worst case all $v' - y$ terms in $v(y+1), \ldots, v(v')$ are 1. Therefore, $p(v')v' \le p(y)y + v' - y$, which is the second line.

The lower bound: the first line holds because of weak monotonicity. The fourth line is again because $p(y)y \le p(v')v'$ when $v' \ge y$. In the middle, when $y - \lfloor mp(y) \rfloor \le v' < y$, at most all $y - v'$ items $v(t_{v'+1}), \ldots, v(t_y)$ are 1. Hence, $p(v')v' + y - v' \ge p(y)y$, which gives us the third line. When $y - m \le v' < y - \lfloor mp(y) \rfloor$, again, because $p(y-m) \ge p(y)$, at most $p(y)$ fraction of the elements $v(t_{y-m+1}), \ldots, v(t_y)$ can be 1. This implies that $p(v')v' + \lfloor mp(y) \rfloor \ge p(y)y$. Rearranging terms, we get the second line.

The envelope is tight because we can construct sequences of $v(t_j)$ which match the upper and lower bound exactly.

B.3 Proof of Theorem 4

The number of calls to QUERY is one plus the number of intervals $(p(l), p(r))$ which enter the stopping condition of function PR. Notice that the boundary (l, r) of every interval $(p(l), p(r))$ which enters the stopping condition must satisfy $r/l \ge 1 + \epsilon$, since otherwise its parent function call already satisfies the stopping condition $r/l \le (1+\epsilon)^2$. Aggregated across all intervals, this can happen at most $\log_{1+\epsilon}(N/\tilde{l})$ times, as claimed.

B.4 Proof of Lemma 3

Let $\tilde{l}' = \max\left\{\lceil \frac{m}{1+\epsilon} \rceil, \tilde{r}\right\}$. Using the full characterization from Theorem 2, we can prove that for $r > l > \tilde{l}'$, $p(r) \le (1+\epsilon)p(l)$.

Because of the definition of $q_1, \ldots, q_K$, we know that $q_1 \le q_2 \le \ldots \le q_K$. Moreover, we know that for $q_i < j < q_{i+1}$, $p(j) \ge p(q_i)/(1+\epsilon)$. Because of the statement in the previous paragraph, we know that $p(j) \le p(q_i)(1+\epsilon)$. Proof completes.

B.5 Proof of Lemma 4

First, the function $p(r)r$ monotonically increases, because for $l < r$, we have

$$r\,p(r) = \sum_{i=1}^{r} v(t_i) \geq \sum_{i=1}^{l} v(t_i) = l\,p(l).$$

Therefore, $p(j)j$ is sandwiched between $p(s_i)s_i$ and $p(s_{i+1})s_{i+1}$ whenever $j \in \{s_i, \ldots, s_{i+1}\}$. Because of the definition of s_{i+1}, for any $j \in \{s_i, \ldots, s_{i+1}\}$, we have $p(j)j \geq p(s_i)s_i$ and $p(j)j \leq (1+\epsilon)p(s_i)s_i$, which finishes the proof.

B.6 Proof of Theorem 6

Suppose the "optimal" algorithm calls QUERY at points $r_1, r_2, \ldots, r_{\text{OPT}}$. We begin with some relevant notation. These points split the entire range between $\tilde{l}$ and N into OPT + 1 *segments*. We call one segment of this type an *opt-segment*. We also define the following tuple: $\langle p(l), p(r), d, \text{SMALL/BIG}\rangle$, where $(p(l), p(r))$ is an interval on which the function PR called during the execution of ADASTRAT, d is the depth of this recursive call, and the fourth entry is either SMALL or BIG. It is BIG if and only if the interval $[l, r]$ covers at least one complete opt-segment. The log distance of a tuple is measured as $\log_{1+\epsilon} r/l$. Two tuples are treated as *cousins* if they are called by a single PR function as two children calls (i.e., one is $(p(l), p(c))$, while the other one is $(p(c), p(r))$). We can *merge* two cousin tuples. The result of merging is the tuple representing the parent function that called the functions represented by these two cousin tuples. Notice that the log distance of the merged tuple is 2 times that of one cousin tuple.

We start with the set Φ, made of tuples $\langle p(l), p(r), d, \text{SMALL/BIG}\rangle$ such that $(p(l), p(r))$ enters the stopping condition of function PR during the execution of the algorithm. We repeat the following *merge* operation on tuples in Φ until no tuple in Φ is tagged with SMALL: (i) Choose a tuple from Φ that has the largest depth d among those tagged with SMALL. (ii) Merge this tuple with its cousin tuple. (iii) Add the merged tuple back into Φ. We refer to the set Φ obtained after merging all SMALL tuples as Φ^E.

First, all tuples in Φ^E are tagged BIG. Therefore, each of them contains an opt-segment. Second, since the tuples in Φ^E are still from the PR algorithm, these tuples must be non-overlapping. Based on these two observations, the number of tuples in Φ^E is bounded by the number of opt-segments, which is OPT+1. How many merge operation could each tuple in Φ^E have gone through? This must be bounded by $\mathrm{O}(\log_2 \log_{1+\epsilon} N)$, since the largest possible log distance is $\log_{1+\epsilon} N$ (the entire range), and each merge operation doubles the log distance.

Now we count the number of QUERY calls that algorithm ADASTRAT makes. It is easy to see that

$$\#\text{QUERY} = |\Phi^E| + \#\text{MERGE} + 1,$$

where #MERGE is the total number of merge operations performed. This number is bounded by

$$|\Phi^E| + |\Phi^E| \cdot \max\{\#\text{MERGE for one tuple in } \Phi^E\} + 1.$$

Combined with the fact that $|\Phi^E| \leq \text{OPT} + 1$ and #MERGE for one tuple in $\Phi^E \leq \log_2 \log_{1+\epsilon} N$, we obtain the claimed bound.

B.7 Detailed Construction of F to Prove Theorem 7

Our proof works by showing that any algorithm that can identify an unknown function from F, which includes $\mathcal{A}$, must make at least J queries. To demonstrate this, our function family F is constructed such that for one separating point, there are at least two functions from F that are more than $(1+\epsilon)^2$ apart at the point. Therefore, $\mathcal{A}$ has to query all J separating points to identify one function from F.

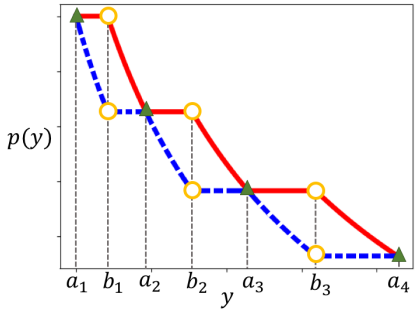

Figure 5: An illustration of the function family supporting the lower bound in Theorem 7.

Specifically, we identify $J+1$ points $a_1, \ldots, a_{J+1}$, where the space between a_j and a_{j+1} is called the j-th interval ($j = 1, \ldots, J$), which is shown as the space between two green triangles in Figure 5. One function f from F either follows the red or the blue curve in one interval. Because we have J intervals in total, this gives us 2^J functions in F. The separating points are shown in yellow circles in Figure 5. We construct these yellow circles to guarantee that the gap (height) is more than $(1+\epsilon)^2$.

We argue that Algorithm $\mathcal{A}$ must make at least one query in each of the J intervals. This is because even if $\mathcal{A}$ knows the exact function segment (red or blue) everywhere except for the j-th interval, it still cannot decide if

f follows the red or the blue curve within the j-th interval.

The exact locations of separating points are as follows. Let $a_j = \lfloor (1+\epsilon')^{4j-4} y_0 \rfloor$, $b_j = \lfloor (1+\epsilon')^{4j-2} y_0 \rfloor$, $(j = 1, \ldots, J+1)$. $N = a_{J+1} = \lfloor (1+\epsilon')^{4J} y_0 \rfloor$. Here, y_0 is a sufficiently large number. Let $1 < \gamma < {}^{(1+\epsilon')}/_{(1+\epsilon)}$. Select 2^J $(\tilde{r}, m)$-weak monotonic functions $f_0, f_1, \ldots, f_{2^J-1}$ such that their values at a_j (shown as the green triangles of Figure 5) are close. More specifically, for all functions $f_u \in \{f_0, f_1, \ldots, f_{2^J-1}\}$, their values at a_j are bounded:

$$\frac{1}{\gamma(1+\epsilon')^{2j-2}} < f_u(a_j) < \frac{\gamma}{(1+\epsilon')^{2j-2}}.$$

For a specific b_j (shown as yellow circles in Figure 5), half of the functions in $\{f_0, f_1, \ldots, f_{2^J-1}\}$ match approximately around a particular value (i.e., follow the red curve). The other half match around a different value (i.e., follow the blue curve). More specifically, for all $u \in \{0, 1, \ldots, 2^J - 1\}$, whose j-th digit of its binary representation is 0, f_u's value at b_j is bounded by

$$\frac{1}{\gamma(1+\epsilon')^{2j-2}} < f_u(b_j) < \frac{\gamma}{(1+\epsilon')^{2j-2}},$$

For all $u \in \{0, 1, \ldots, 2^J - 1\}$, whose j-th digit of its binary representation is 1, f_u's value at b_j is bounded by

$$\frac{1}{\gamma(1+\epsilon')^{2j}} < f_u(b_j) < \frac{\gamma}{(1+\epsilon')^{2j}},$$

Suppose algorithm $\mathcal{A}$ outputs an $(1+\epsilon)$-approximate curve for any $(\tilde{r}, m)$-weak monotonic precision function. Starting with one unknown function f drawn from the set $\{f_0, f_1, \ldots, f_{2^J-1}\}$, we can use algorithm $\mathcal{A}$ to identify which function f actually is. To see this, we examine the approximate values $\tilde{f}(b_1), \ldots, \tilde{f}(b_J)$ returned by algorithm $\mathcal{A}$. If

$$\tilde{f}(b_j) > \frac{1}{(1+\epsilon')^{2j-1}},$$

we know that

$$\frac{1}{\gamma(1+\epsilon')^{2j-2}} < f(b_j) < \frac{\gamma}{(1+\epsilon')^{2j-2}},$$

because otherwise, $\tilde{f}(b_j)$ and $f(b_j)$ are more than $(1+\epsilon)^2$ apart. The other side of the argument also holds.

B.8 Proof of Theorem 8

First, assuming that we have access to $r_1, \ldots, r_K$, we access the values of the first $\tilde{l}$ terms $v(t_1), \ldots, v(t_{\tilde{l}})$. We then randomly access $\lceil \frac{r_1 - \tilde{l}}{r_1} \rceil s$ items from $\tilde{v}(t_{\tilde{l}+1}), \ldots, \tilde{v}(t_{r_1})$, randomly access $\lceil \frac{r_2 - r_1}{r_2} \rceil s$ items from $\tilde{v}(t_{r_1+1}), \ldots, \tilde{v}(t_{r_2})$, and so on, until we randomly access $\lceil \frac{r_K - r_{K-1}}{r_K} \rceil s$ items from $\tilde{v}(t_{r_{K-1}+1}), \ldots, \tilde{v}(t_{r_K})$.

When we compute $\textsc{Query}(r_i, T, \tilde{v})$, we randomly subsample accessed items in the range of $1, \ldots, \tilde{l}$ with probability $\frac{r_i - r_{i-1}}{r_i}$; randomly subsample accessed items in the range of $\tilde{l}+1, \ldots, r_1$ with probability $\frac{r_i - r_{i-1}}{r_i} \frac{r_1}{r_1 - \tilde{l}}$; randomly subsample accessed items in the range of $r_1 + 1, \ldots, r_2$ with probability $\frac{r_i - r_{i-1}}{r_i} \frac{r_2}{r_2 - r_1}$; and so on; randomly subsample accessed items in the range of $r_{i-2} + 1, \ldots, r_{i-1}$ with probability $\frac{r_i - r_{i-1}}{r_i} \frac{r_{i-1}}{r_{i-1} - r_{i-2}}$; and take all accessed items in the range of $r_{i-1} + 1, \ldots, r_i$. We use the empirical mean of these subsampled items to compute the estimation $\tilde{p}(r_i)$. Using Hoeffding's inequality, Eq. (2), we know that the estimation is a β-approximation with confidence at least $1 - \delta/K$.

Next, applying the union bound over all $i \in \{1, \ldots, K\}$, we have overall confidence of at least $1 - \delta$ that the estimations are β-approximations simultaneously for all K points $r_1, \ldots, r_K$.

Nevertheless, we do not know the location of $r_1, \ldots, r_K$ upfront. We thus take an iterative deepening style adaptive sampling scheme, as detailed in the last two paragraphs of Section 4. In particular, we gradually increase the number of samples in each interval as we increase K; and we sample more points for intervals that fall below the sample density thresholds when we introduce new query points.