On Active Learning of Record Matching Packages

Arvind Arasu
Microsoft Research
Redmond, WA 98052, USA
arvinda@microsoft.com

Michaela Götz
Cornell University
Ithaca, NY 14853, USA
goetz@cs.cornell.edu

Raghav Kaushik
Microsoft Research
Redmond, WA 98052, USA
skaushi@microsoft.com

ABSTRACT

We consider the problem of learning a record matching package (classifier) in an active learning setting. In active learning, the learning algorithm picks the set of examples to be labeled, unlike more traditional passive learning setting where a user selects the labeled examples. Active learning is important for record matching since manually identifying a suitable set of labeled examples is difficult. Previous algorithms that use active learning for record matching have serious limitations: The packages that they learn lack quality guarantees and the algorithms do not scale to large input sizes. We present new algorithms for this problem that overcome these limitations. Our algorithms are fundamentally different from traditional active learning approaches, and are designed ground up to exploit problem characteristics specific to record matching. We include a detailed experimental evaluation on real-world data demonstrating the effectiveness of our algorithms.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

General Terms

Algorithms, Performance

Keywords

Data Cleaning, Record Matching, Active Learning

1. INTRODUCTION

Record Matching is the problem of identifying matching or duplicate records, records that correspond to the same real-world entity. An example record matching task is to identify bibliographic records in Citeseer [13] and DBLP [18] that correspond to the same publication. Figure 1 shows a toy example with two tables R (records R1-R3) and S (records S1-S3) containing organization names and addresses, and the goal is to find pairs of records that represent the same organization; these are likely to be the pairs (R1, S3) and (R2, S2). Record matching is a well-studied problem and has applications in information integration [5, 22], data warehousing [1], census data [42] and health-care records management [35].

ID	*Name*	*Street*	*City*	*Phone*
R1	SWI Alloys Inc	456 Medical Dr	Albany	5181234567
R2	ABC Rural Telephone	1234 Market St	Fremont	3179876543
R3	Bank of New York	1 E 31st St	New York	2120001111
S1	First Tech	156th Ave	Boise	
S2	ABC Cellular	PO Box 9862	Fremont	3179876543
S3	SWI Alloys	456 Medical Dr	Albany	

Figure 1: An example record matching task

The standard approach to record matching is to use textual similarity between the records to determine whether or not two records are matches [20]. Informally, in Figure 1, the matching pair (R1, S3) is textually similar, while the non-matching pair (R1, S1) is not. Current approaches typically compute a variety of *similarity scores* for a candidate pair of records and these scores are combined using some logic to determine if the pair is a match or not. A similarity score quantifies textual similarity between the two records on some subset of attributes, and is computed using a *string similarity function* such as edit distance, jaccard, and cosine similarity [20].

Since manually coming up with a logic for combining similarity scores is difficult, the current state-of-art uses a *learning based* approach: In this approach, record matching is viewed as a classification problem where each pair has to be classified as a match or a non-match, and a suitable classifier is learned using *labeled examples* of matching and non-matching pairs. The pair-wise similarity scores serve as features for the classification. Prior work has considered a variety of classifiers such as SVMs [7], decision trees [11, 37], and naive Bayes [42].

However, there is a crucial difference between record matching and standard classification problems: In a typical record matching task, the number of non-matches far exceeds the number of matches. For concreteness, consider a record matching task involving two tables with a million records each, and assume that each record matches with 10 records of the other table on average. The number of matches for this task is $\approx 10^7$, while the number of non-matches is $\approx 10^{12}$. This imbalance makes it very difficult to identify a suitable set of labeled examples so that a learned classifier has high quality [37]. Standard techniques such as picking pairs of records at random or using some other distribution do not work very well. Prior work [8, 11] has proposed using a *filter* based on textual similarity to eliminate a large number of non-matches followed by sampling to pick examples. As we show using our experiments, while this approach mitigates the problem described above, it does not eliminate it. It introduces the additional problem of picking a good filter.

Active Learning: Motivated by these considerations, we explore the use of *active learning* for record matching. In active learning, the learning algorithm itself picks the examples to be labeled. The hope is that the algorithm can exploit this additional flexibility to pick examples that are most informative for the learning task. This eliminates the user's burden of picking suitable examples or a good filter.

While active learning has been previously studied for record matching [37, 40], the proposed algorithms have two limitations: First, these algorithms do not provide a principled interface using which a user can control the quality of a learned classifier. (Informally, in record matching, the quality of a classifier is measured using its *precision* and *recall*; the recall of a classifier is the number of pairs that it classifies as a match and the precision is the fraction of these pairs that are *true* matches.) For example, we do not know of a systematic way of using the algorithms of [37, 40] to ensure that the learned classifier has precision above some threshold. Further, we observed in our experiments that the behavior of these algorithms can be unpredictable and precision and/or recall of the learned classifier can decrease when more labeled examples are provided. This unpredictability makes it difficult to use these algorithms in record matching settings with specific quality requirements. A second limitation of these algorithms is that they do not scale to large inputs. For each requested label, these algorithms iterate over all record pairs, and the number of such pairs is quadratic in the input size. However, we can address this limitation using *blocking* techniques as we discuss subsequently.

We propose new algorithms for active learning of record matching classifiers. Our algorithms cover decision trees and linear classifiers (which include SVMs), and do not have the limitations of previous algorithms: In particular, our algorithms allow a user to specify a precision threshold as input. The learned classifier is guaranteed to have a precision greater than this threshold and (under certain reasonable assumptions) recall close to the best possible given the precision constraint. Our algorithms also differ from previous algorithms in that they are designed from scratch for record matching, and do not just invoke a known learning algorithm as a black-box. This enables our algorithms to exploit problem features that are specific to record matching. Our active learning algorithms can also be easily adapted to yield new learning algorithms in the traditional passive learning setting.

Efficiency considerations: For large record matching instances, it is inefficient to consider all pairs of candidate records, classify them as a match or non-match, and output those classified as a match. The traditional approach to handling this problem is to use *blocking* [6, 32]. Blocking is a heuristic filtering step that selects a subset of candidate pairs of records, and only the selected pairs are considered for subsequent classification. A good blocking scheme has an efficient implementation and eliminates few true matches. An example blocking scheme for the instances like Figure 1 is to select pairs of records that agree on the first letter of the *Name* column for subsequent classification. A more sophisticated blocking scheme is to select pairs that have jaccard similarity at least 0.8 on the *Name* column; such *string-similarity joins* can be efficiently evaluated using techniques proposed in [2, 25, 38].

We develop a simple integration of blocking into the learning problem that enables our algorithms to handle large inputs. Given a blocking scheme, our algorithms learn a classifier that when used in conjunction with the blocking scheme, has maximum recall and precision above a specified threshold. This integration of blocking and active learning reduces the number of labeling requests. For example, assume that the first letter blocking scheme described above is used for record matching. An active learning algorithm without a knowledge of this blocking scheme might request a label for a pair that does not agree on the first letter, and the labeling effort on that pair is wasted since such pairs would never be considered for classification. We argue that previous algorithms [37, 40] can also be modified to similarly exploit blocking functions. In our experiments, we use these algorithms with this modification for comparison purposes (and demonstrate that our algorithms perform better).

Roadmap: In Section 2, we introduce definitions and notation and formalize the active learning problem. In Section 3, we introduce and discuss an interesting *monotonicity* property in record matching, which is exploited in the design of our algorithms. We present our algorithms in Sections 4 and 5, and evaluate their performance empirically in Section 6. We cover related work in Section 7.

2. PRELIMINARIES

The record matching problem is the problem of identifying all pairs of matching records $(r, s) \in R \times S$, given two sets of input records, R and S. Two records match if they represent the same real-world entity. This notion of a match lacks a precise characterization; a human judge would typically use a variety of semantic cues to determine if two records match or not.

Our overall goal is to learn a *record matching package* for inputs R and S. A record matching package for R and S is a program that performs record matching over them, i.e., its desired output is the set of all matching pairs $(r, s) \in R \times S$. Since record matching is an informally stated task, it is difficult to learn a "perfect" record matching package that produces exactly the desired output, so our goal is to learn a package that closely approximates the ideal output.

The degree of approximation is typically measured using two well-known statistics: *precision* and *recall* [7, 37]. The *precision* of a package is the fraction of predicted matches (pairs in its output) that are true matches. The *recall* of a package is the number of predicted matches, i.e., its output size. Our definition differs from the classical definition from information retrieval [4]: the fraction of true matches that are also predicted as matches by a package. Our definition is simpler to calculate, while retaining the utility of the classical definition for purposes of comparing different packages. Given a record matching package, we can easily compute its precision and recall. The recall of the package is simply its output size. The precision of the package can be estimated by labeling a random sample of output records.

We seek record matching packages with high precision and recall. However, maximizing precision and maximizing recall are conflicting goals: we can increase precision at the cost of recall and vice-versa. For example, the record matching package that outputs the entire $(R \times S)$ has high recall but is likely to have low precision. Similarly, the package that outputs only identical records as matches is likely to have high precision, but low recall. The above discussion raises the question of how we define the "best" package. In this paper, we seek a record matching package that maximizes recall while ensuring that the precision is at least τ for some input threshold $\tau \in [0, 1]$. Prior work [7, 37] has considered other definitions for the best package, such as the one that maximizes the *F-measure*.

An alternative quality measure would be the *misclassification rate*, defined as the fraction of record pairs that are incorrectly classified by the package. This quality metric is standard for many classification problems but unsuitable for record matching, since, typically, the number of non-matches far exceeds the number of matches [37]. Therefore, a package that outputs an empty set implicitly classifying all pairs as non-matches has very low misclassification rate, but is not very useful.

Similarity Space

As mentioned in Section 1, record matching packages use textual similarity between two records to decide if they are matches or not. Formally, we assume there exist d predefined *similarity dimensions* for a given record matching task: $\bar{F} = F_1, \ldots, F_d$. A similarity dimension measures the similarity between an attribute of R and an attribute of S using a similarity function. Without loss of generality, we assume that all similarity functions return values in the range $[0, 1]$. We can map every pair $(r, s) \in R \times S$ to a d-dimensional similarity vector $\langle f_1, \ldots, f_d \rangle \in [0, 1]^d$, which we denote $\bar{F}(r, s)$; f_i is the similarity between r and s on dimension F_i. (In machine learning terminology, $\bar{F}(r, s)$ is the feature vector which forms the basis for classifying the pair as a match or non-match.) We call the space of vectors in $[0, 1]^d$ the *similarity space*.

Example 1. Consider a hypothetical record matching task involving tables ORG-R and ORG-S. Both tables have the same schema: (*Name*, *Street*, *City*, *Phone*), and contain organization records. An example similarity dimension is *Jaccard*(*Name*), which represents jaccard similarity on the *Name* attribute. The same attribute can be used with more than one similarity function: e.g., we can have *Edit*(*Name*) and *Jaccard*(*Name*) as two dimensions. □

Record Matching Package Classes

A record matching package $\mathcal{M}$ is conceptually a classifier that classifies a record pair as a match or a non-match based on their similarity vector. Formally, $\mathcal{M}$ is a binary function with signature $[0, 1]^d \rightarrow \{true, false\}$. A pair $(r, s) \in R \times S$ is classified as a match if $\mathcal{M}(\bar{F}(r, s)) = true$ and a non-match if $\mathcal{M}(\bar{F}(r, s)) = false$. In the following, we shorten $\mathcal{M}(\bar{F}(r, s))$ to $\mathcal{M}(r, s)$.

Two popular and well-studied classifiers for record matching are SVMs [9] and decision trees [36]. Prior work [37] has shown that other common classifiers such as naive Bayes [33] are less suited for record matching compared to SVMs and decision trees. We now define two classes of binary functions, *threshold-based boolean function* and *linear classifiers* that subsume decision trees and SVMs. In particular, threshold-based boolean functions are generalizations of decision trees, while SVMs (without the kernel trick [9]) are instances of linear classifiers.

Definition 1. A threshold-based boolean function (hereafter, a *threshold function*) is a boolean formula whose basic propositions are of the form $(F_i \geq \theta)$. For a similarity vector $f = \langle f_1, \ldots, f_d \rangle$, the predicate $(F_i \geq \theta)$ evaluates to *true iff* $f_i \geq \theta$.

Definition 2. A linear classifier L is of the form $\sum_i w_i F_i \geq 1$, where w_i, $i \in [1, d]$ are real numbers. L evaluates a similarity vector $f = \langle f_1, \ldots, f_d \rangle$ to *true iff* $\sum_i w_i f_i \geq 1$.

Example 2. For the record matching task of Example 1, an example threshold function is $(Jaccard(Name) \geq 0.7) \wedge (Jaccard(Street) \geq 0.6) \vee (Equality(Phone) = 1.0)$. An example of a linear classifier is $(0.5 \cdot Jaccard(Name) + 0.7 \cdot Jaccard(Street) \geq 1.0)$. □

Basic Problem Formulation

We can now state the problem of learning a record matching package as follows: Given two sets of input records R and S, a set of predefined similarity dimensions $\bar{F} = F_1, \ldots, F_d$ over schema of R and S, and a precision threshold τ, learn a record matching package (belonging to one of the two classes above) with precision at least τ that maximizes recall. The learning algorithm has access to a human *labeler*, who can label selected pairs $(r, s) \in R \times S$ as a match or a non-match.

An algorithm for the learning problem has two associated costs—*labeling cost* and *computational cost*—that we seek to minimize. The labeling cost is the number of examples for which it requests labels and the computational cost is the time it takes to produce its output. We believe that labeling is a more expensive resource, so (informally) we seek to minimize labeling cost, while keeping the computational cost within reasonable limits.

Efficiency Considerations

For large inputs R and S, it is impractical to enumerate all pairs $(r, s) \in R \times S$, classify them using a learned classifier $\mathcal{M}$, and output the matches. As mentioned in Section 1, the traditional approach to scaling record matching to large inputs involves the use of blocking [6, 32] as a pre-filtering step. A blocking scheme quickly identifies a relatively small subset of record pairs, and only these pairs are considered for subsequent classification. Formally, we define a blocking scheme as a binary function $\mathcal{B}$: $R \times S \rightarrow \{true, false\}$ with the property that the set of all $(r, s) \in R \times S$ such that $\mathcal{B}(r, s) = true$ can be efficiently computed. We note that our definition of a blocking function is fairly general. In particular, a blocking function can be a string similarity predicate such as $(Jaccard \geq \theta)$ and, more generally, a disjunction of a small number of such predicates. When a blocking function $\mathcal{B}$ is used in conjunction with classifier $\mathcal{M}$, the end-to-end record matching package corresponds to the binary function $(\mathcal{B} \wedge \mathcal{M})$.

We integrate blocking into the learning problem by providing the learning algorithm knowledge of the blocking function $\mathcal{B}$. Formally, the modified learning problem is the following: Given R, S, $\bar{F}$, and τ as before, and a blocking function $\mathcal{B}$, identify a binary function $\mathcal{M}$ such that the precision of $(\mathcal{B} \wedge \mathcal{M})$ is at least τ and the recall of $(\mathcal{B} \wedge \mathcal{M})$ is maximized.

There are two advantages to integrating blocking with learning. First, it allows the learning algorithm to scale to large inputs; in fact, previous active learning algorithms [37, 40] can also be modified to exploit blocking and scale to larger inputs. Second, the integration eliminates inefficiencies in learning since the learning algorithm can avoid seeking labels for pairs (r, s) such that $\mathcal{B}(r, s) = false$.

3. MONOTONICITY OF PRECISION

Informally, we expect a pair of records that is textually similar to be more likely a match than a pair that is not. As we will see, this simple observation can be exploited while learning record matching packages. We formalize this observation, which we call the *monotonicity of precision*, next.

We define a partial ordering $(\preceq)$ on points in similarity space. Let $f = \langle f_1, \ldots f_d \rangle$ and $g = \langle g_1, \ldots, g_d \rangle$ be two points in $[0, 1]^d$. We say that g *dominates* f, denoted $g \succeq f$ (equivalently, $f \preceq g$) if $f_i \leq g_i$ for all $1 \leq i \leq d$. If $f \preceq g$ and $f_i \neq g_i$ for some $1 \leq i \leq d$, we denote $f \prec g$ (equivalently, $g \succ f$).

A *closed region* $\mathcal{C}$ in similarity space is a set of points with the property that any two points in $\mathcal{C}$ can be connected by a curve that lies wholly within $\mathcal{C}$. We say that a closed region $\mathcal{C}_1$ dominates a closed region $\mathcal{C}_2$, denoted $\mathcal{C}_1 \succeq \mathcal{C}_2$, if every point in $\mathcal{C}_1$ dominates every point in $\mathcal{C}_2$. Figure 2(a) shows two closed regions $\mathcal{C}_1 \succeq \mathcal{C}_2$ in a two-dimensional similarity space. Given input tables R and S, we define the precision of a closed region $\mathcal{C}$, denoted $Prec(\mathcal{C})$ to be the fraction of matches among pairs $(r, s) \in R \times S$ that belong to $\mathcal{C}$, i.e., $\bar{F}(r, s) \in \mathcal{C}$. Similarly, we define recall of $\mathcal{C}$, denoted $Recall(\mathcal{C})$ to be the number of pairs $(r, s) \in R \times S$ that belong to $\mathcal{C}$.

Definition 3. Given input tables R and S, a set of similarity dimensions $\bar{F} = F_1, \ldots, F_d$, we say that precision is monotonic with respect to similarity if for any two closed regions $\mathcal{C}_1 \succeq \mathcal{C}_2$ in $[0, 1]^d$, $Prec(\mathcal{C}_1) \geq Prec(\mathcal{C}_2)$. If precision is monotonic with respect to similarity, we say that the monotonicity assumption holds.

If the monotonicity assumption is valid, then the precision of region $\mathcal{C}_1$ is higher than the precision of region $\mathcal{C}_2$ in Figure 2(a).

This monotonicity assumption while intuitive is not universally valid. We can easily construct a set of meaningless similarity dimensions for which the precision is not monotonic. Even for standard similarity functions and real-world record matching tasks, the monotonicity assumption is not valid at the record level, since we can usually find a non-matching record pair whose similarity vector dominates that of a matching record pair.

However, in practice, the monotonicity assumption generally holds when the recall of $\mathcal{C}_1$ and $\mathcal{C}_2$ is high. In other words, the monotonicity assumption holds in "aggregation" when both $\mathcal{C}_1$ and $\mathcal{C}_2$ contain a large number of record pairs $(r, s) \in R \times S$. Figures 2(b) and 2(c) illustrate the monotonicity assumption for two real-world datasets . The details of the datasets are provided in Section 6. Figure 2(b) uses the publication domain. It plots the precision of five non-overlapping regions, where each region contains points with jaccard similarity (over citation strings) in some range. For example, the region corresponding to the range $[0.38, 0.40]$ dominates the region corresponding to the range $[0.36, 0.38]$, and we see a decrease in precision. Figure 2(c) uses the organization domain, and the regions are now defined by considering the average of jaccard over name and jaccard over address strings. Again, we can verify that the monotonicity assumption holds for these regions. We have empirically verified the monotonicity assumption for a variety of other datasets; some of these results are included in the full version of the paper.

Implications for Learning Record Matching Packages

When precision is monotonic with respect to similarity, we can restrict ourselves to a simpler class of binary functions called *monotonic* binary functions (formalized in Theorem 1). A binary function $\mathcal{M}$ is monotonic if for any two similarity vectors $f \succeq g$, $\mathcal{M}(g) = true \Rightarrow \mathcal{M}(f) = true$. A threshold function is monotonic if it does not involve any negations and a linear classifier $\sum_i w_i F_i \geq 1$ is monotonic if each $w_i \geq 0$.

THEOREM 1. *For any record matching task for which monotonicity assumption holds, if there exists a binary function $\mathcal{M}$ with precision $\geq \tau$ and recall r, then there exists a monotonic binary function $\mathcal{M}'$ with precision $\geq \tau$ and recall $\geq r$.*

The above statement holds even if we consider exclusively threshold functions or linear classifiers. In the full version of the paper, we formally show that even when monotonicity of precision holds only approximately, we can restrict ourselves to monotonic functions, without sacrificing much quality.

In the following, we use the above observations to restrict ourselves to monotonic binary functions. Further, our learning algorithms are designed assuming precision is monotonic. Designing algorithms assuming monotonicity might seem *ad hoc* given that monotonicity is not universally valid (although as we discussed earlier it is generally valid in an aggregate sense in practice). We argue that whether or not monotonicity is completely valid, we can view it as a useful trade-off between learning efficiency and quality. We might possibly miss the best package by (incorrectly) assuming monotonicity, but the assumption makes the learning problem more tractable. The monotonicity assumption is similar in spirit to the independence assumption in naive Bayes classifier [33]: in practice words in documents are strongly correlated, but we can learn a useful classifier by assuming words to be independent. In Section 6, we also provide empirical justification for designing our algorithms around monotonicity, demonstrating that our algorithms perform significantly better than state-of-art algorithms which do not assume monotonicity.

4. ALGORITHMS

We now present our algorithms for learning record matching packages. For simplicity of exposition, we present our algorithms for the basic learning problem without a blocking function. Exploiting blocking functions is discussed in Section 5.

4.1 Conjunction of Similarity Thresholds

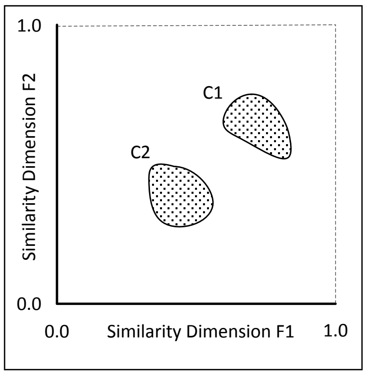
We now consider a simple class of threshold functions obtained by conjunction of threshold predicates. Without loss of generality, a function $\mathcal{M}$ belonging to this class is of the form $(F_1 \geq \theta_1) \wedge \cdots \wedge (F_d \geq \theta_d)$. There exists a one-one correspondence between functions belonging to this class and points in the similarity space. The function $\mathcal{M}$ above corresponds to the point $p = \langle \theta_1, \ldots, \theta_d \rangle$, and note that for any $f \in [0, 1]^d$, $\mathcal{M}(f) = true$ *iff* $p \preceq f$. We define precision and recall of a point p (denoted $Prec(p)$ and $Recall(p)$) to be the precision and recall of its corresponding function. The problem of finding a function $\mathcal{M}$ with maximum recall such that $Prec(\mathcal{M}) \geq \tau$ is therefore equivalent to the problem of finding a point $p \in [0, 1]^d$ with maximum recall and precision $\geq \tau$.

The naive algorithm enumerates all points $p \in [0, 1]^d$ and calculates their precision and recall. Among all points p with $Prec(p) \geq \tau$, it picks as its output the point with maximum recall.

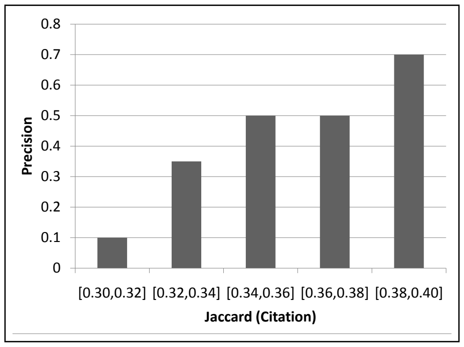
The naive algorithm is infeasible since there are an infinite number of points in $p \in [0, 1]^d$. We can use a simple approximation trick to make the number of points finite[1]: We fix a integer value k, called the *granularity parameter*. We define a special set of $(k + 1)^d$ points, called *points at granularity* k, to be the set of all points of the form $p = \langle p_1, \ldots, p_d \rangle$, where each p_i is of the form j/k, $j \in \{0, 1, \ldots, k\}$. If we partition the similarity space into k^d identical cells with sides $(1/k)$, then the vertices of the cells correspond to the points at granularity k. Figure 3(a) illustrates this concept for $k = 10$ and $d = 2$. Instead of considering all points in $[0, 1]^d$, the modified naive algorithm only considers points at granularity k, and outputs the point with maximum recall subject to the precision constraint. If M denotes the maximum number of pairs in $(R \times S)$ that belong to any single cell, then this algorithm picks a point whose recall is at most M away from the optimal recall.

To implement this algorithm, we need to compute precision and recall of various points. In this section, we develop our algorithms assuming two *oracles* that compute precision and recall of a binary

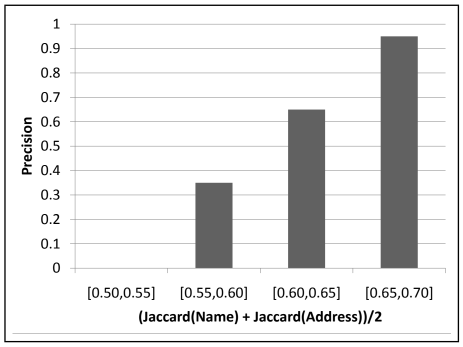
[1]The number of points can also be made finite by only considering points $\bar{F}(r, s)$, $(r, s) \in R \times S$. However, enumerating these points is expensive as well.



(a) Closed regions in similarity space



(b) Monotonicity for publication data



(c) Monotonicity for organization data

Figure 2: Monotonicity Assumption

function (recall that a point is an instance of a binary function). We analyze the cost of our algorithms using the number of calls to these oracles that the algorithms make. In Section 5, we discuss the implementation of these oracles. As we will see, computing precision requires human labeling while computing recall does not. We therefore primarily seek to minimize the number of calls to the precision oracle.

4.1.1 Exploiting Monotonicity of Precision

We now improve upon the naive algorithm by exploiting monotonicity of precision with similarity. Consider two points $p_1 \preceq p_2$ such that $Prec(p_1) \geq \tau$. If we assume monotonicity of precision, we can prove that $Prec(p_2) \geq Prec(p_1) \geq \tau$. We also observe that recall has an *anti-monotonic property*: if $p_1 \preceq p_2$ then $Recall(p_1) \geq Recall(p_2)$. This property follows from the definition of recall and, unlike monotonicity of precision, is unconditionally true. The above discussion suggests that we can remove from consideration points such as p_2 which dominate another high-precision point. In other words, it is sufficient to consider points p that are "minimally precise," meaning any point $p' \prec p$ does *not* satisfy the precision constraint. We formalize this notion with respect to an arbitrary monotonic binary predicate[2] $Pred$.

Definition 4. Given a monotone binary predicate $Pred$ defined over points in $[0,1]^d$, we say a point $p \in [0,1]^d$ is *minimally true* (or $MinTrue$) if $Pred(p) = true$ and $\forall p' \prec p\ Pred(p') = false$. We denote using $MinTrue(Pred)$ the set of all minimally true points for $Pred$.

The dual notion of *maximally false* is also useful:

Definition 5. Given a monotone binary predicate $Pred$ defined over points in $[0,1]^d$, we say a point $p \in [0,1]^d$ is *maximally false* (or $MaxFalse$) if $Pred(p) = false$ and $\forall p' \succ p\ Pred(p') = true$. We denote using $MaxFalse(Pred)$ the set of all maximally false points for $Pred$.

We call a point p a *boundary point* if it is either minimally true or maximally false. When considering points at some granularity k, the universal quantifier in the above definitions is with respect to points at granularity k.

Example 3. Figure 3(a) illustrates these ideas for $d = 2$. The dotted curved-line is the $Prec() \geq \tau$ boundary; all points above the line have precision $\geq \tau$ and all points below, $< \tau$. The set of all $MinTrue$ points (for $Prec() \geq \tau$) at granularity $k = 10$ are shown using green circles and the set of all $MaxFalse$ points, using red squares. □

[2]A monotonic binary predicate is simply a monotonic binary function:$[0,1]^d \rightarrow \{true, false\}$

Algorithm 1 ALGP: Learning a conjunction of similarity thresholds

```
   Inputs: $\tau$: precision threshold; $k$: granularity parameter
1: $p_{best} \leftarrow null$                         ▷ current best point
2: $r_{best} \leftarrow 0$                            ▷ current best recall
3: for all $p \in MinTrue(Prec(p) \geq \tau)$ do
4:     if $Recall(p) > r_{best}$ then
5:         $p_{best} \leftarrow p$
6:         $r_{best} \leftarrow Recall(p)$
7:     end if
8: end for
9: output $p_{best}$
```

Algorithm ALGP (Algorithm 1) formalizes the discussion above: it enumerates all $MinTrue(Prec() \geq \tau)$ points at granularity k, and outputs the one with best recall.

We next consider the problem of enumerating $MinTrue(Prec() \geq \tau)$. This problem is related to the problem of identifying *maximal frequent itemsets* in data mining [24]. If we view itemsets as binary vectors in some high dimensional space, the maximal frequent itemsets with frequency $\geq T$ are precisely the set of $MaxFalse$ points with respect to the predicate $(Frequency < T)$. (The enumeration of $MinTrue$ and $MaxFalse$ points are dual problems.) Our problem is more general since we deal with non-binary vectors and, accordingly, our algorithm for enumerating $MinTrue$ points is a generalization of the algorithm presented in [24].

Algorithm 2 presents ENUMERATEBOUNDARY, an algorithm for enumerating boundary points ($MinTrue$ and $MaxFalse$) for a general monotonic binary predicate $Pred$. ALGP invokes ENUMERATEBOUNDARY with the predicate $Prec() \geq \tau$. In Section 4.1.2 we present a more sophisticated algorithm for learning conjunction of thresholds that uses ENUMERATEBOUNDARY with a different predicate.

ENUMERATEBOUNDARY maintains the "current" set of minimally true and maximally false points in the variables $MinTrueSet$ and $MaxFalseSet$, respectively. Each iteration of the algorithm adds a new point to either $MinTrueSet$ (Step 8) or $MaxFalseSet$ (Step 12). At all times, the algorithm maintains in variable $MaxCand$ (for maximal candidates), the set of all maximal points p with the property $\forall p_{mt} \in MinTrueSet, p_{mt} \npreceq p$ and $\forall p_{mf} \in$

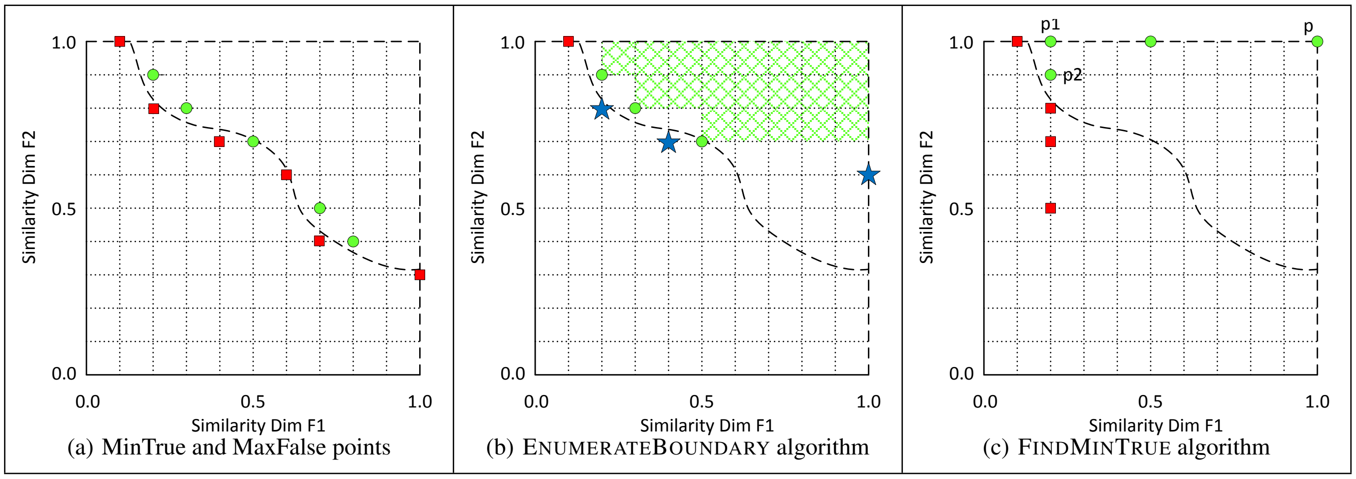


Figure 3: Running example for Section 4.1

$MaxFalseSet, p \not\preceq p_{mf}$. The points p in $MaxCand$ are maximal in the sense that no other point $p' \succ p$ has this property.

In each iteration, ENUMERATEBOUNDARY picks a point p from $MaxCand$ (Step 5). If $Pred(p) = false$, then we can prove that p is a maximally false point, and p is added to $MaxFalseSet$ (Step 12). If $Pred(p) = true$ then, by definition, there exists some minimally true point $p_{mt} \preceq p$, not in the current $MinTrueSet$. ENUMERATEBOUNDARY uses subroutine FINDMINTRUE (Algorithm 3) to find one such point, and adds it to $MintrueSet$ (Step 8). When a new $MinTrue$ point is found, $MaxCand$ needs to be updated to preserve the invariant stated earlier. Due to space constraints we defer the details of procedure UPDATECANDIDATES that updates $MaxCand$ to the full version of the paper.

Example 4. Figure 3(b) illustrates a possible state of ENUMERATEBOUNDARY after four iterations over the running example introduced in Example 3. At this point, $MinTrueSet$ contains three points shown as green circles and $MaxFalseSet$ contains one point shown as a red square. The points in $MaxCand$ are shown as blue stars. □

We next discuss the procedure FINDMINTRUE (Algorithm 3) for finding a minimally true point starting with a true point p. FINDMINTRUE starts with point p and navigates the similarity space to a $MinTrue$ point that p dominates. In particular, FINDMINTRUE navigates through a sequence of points $p = p_0 \rightsquigarrow p_1 \rightsquigarrow \cdots \rightsquigarrow p_d$, and returns p_d as its output. Point p_i agrees with point p_{i-1} on all dimensions except i. For dimension i, p_i contains the smallest value v/k such that $Pred(p_i)$ remains true; the smallest value is found using binary search. We can prove that the final point p_d is minimally true and $p_d \preceq p$.

Example 5. Figure 3(c) illustrates FINDMINTRUE for our running example for the predicate $Prec() \geq \tau$. FINDMINTRUE navigates through the points $p \rightsquigarrow p_1 \rightsquigarrow p_2$, and returns p_2 as a $MinTrue$ point. The points at which the algorithm evaluates precision to check if it is $\geq \tau$ are also shown. □

Analysis of AlgP

The following lemma characterizes the overall performance of ALGP in terms of the number of calls to the precision and recall oracles. These are roughly proportional to the number of boundary points for the predicate $Prec() \geq \tau$.

LEMMA 1. *The number of points p for which* ALGP *evaluates precision $Prec(p)$ is $O(\log k \cdot d \cdot |MinTrue| + |MaxFalse|)$. The number of points p for which it evaluates the recall $Recall(p)$ is $O(|MinTrue|)$.*

We can use Lemma 1 to illustrate the theoretical benefits of active learning over passive learning. A passive learning algorithm cannot request precision and recall values iteratively. Instead it has to request them upfront before having seen the inputs R and S. Any passive learning algorithm essentially has to know the precision and recall of all $(k+1)^d$ points since for any such point there is an input R, S for which this point is the only correct answer. The number of requests made by our active learning algorithm can be much smaller. This illustrates the power of picking examples iteratively. A similar result was obtained for learning a homogeneous linear separator of points uniformly distributed in the unit sphere in which an active learning algorithm performs much better than any passive learning algorithm [17]. The following lemma states that ALGP is close to worst-case optimal, considering active learning algorithms in the precision-recall oracle computation model:

LEMMA 2. *Any algorithm for learning conjunction of similarity thresholds requires $\Omega(|MinTrue|+|MaxFalse|)$ precision evaluations and $\Omega(|MinTrue|)$ recall evaluations in the worst-case.*

While we cannot improve upon ALGP in the worst case, in practice, we can exploit recall to significantly improve performance. We discuss this algorithm next.

4.1.2 Exploiting Recall

Recall that ALGP examines all $MinTrue$ points for the predicate $(Prec \geq \tau)$, and picks the one with maximum recall. In practice, there is a large variation in the recall of $MinTrue$ points: there are typically a few points with high recall and a large number with relatively low recall. This variation arises since the record pairs in $(R \times S)$ are not uniformly distributed in the similarity space. ALGP tends to waste a lot of human labeling effort estimating precision of regions with low recall.

We now present an improved algorithm (ALGPR) for identifying the best conjunction of similarity thresholds that exploits recall information to reduce the labeling cost. ALGPR makes more calls to the recall oracle but fewer to the precision oracle compared to ALGP; note that recall can be computed without labeling.

We call a point a *candidate* if it is a $MinTrue$ point for the predicate $(Prec \geq \tau)$. The basic idea of the algorithm is sim-

Algorithm 2 Enumerate all $MinTrue$ and $MaxFalse$ points

```
Require: $Pred$ is a monotone binary predicate
 1: procedure EnumerateBoundary($Pred$)
 2:     $MinTrueSet = MaxFalseSet = \emptyset$
 3:     $MaxCand = \{\langle 1, \ldots, 1\rangle\}$
 4:     while $MaxCand$ is not empty do
 5:         $p \leftarrow$ some point in $MaxCand$
 6:         $MaxCand \leftarrow MaxCand \setminus p$
 7:         if $Pred(p) = true$ then
 8:             $p_{\min} \leftarrow$ FindMinTrue($p$, $Pred$)
 9:             $MinTrue \leftarrow MinTrue \cup \{p_{\min}\}$
10:             $MaxCand \leftarrow$ UpdateCandidates($MaxCand$, $p_{\min}$, $MinTrue$)
11:         else
12:             $MaxFalse \leftarrow MaxFalse \cup \{p\}$
13:         end if
14:     end while
15:     return $\langle MinTrue, MaxFalse\rangle$
16: end procedure

17: procedure UpdateCandidates($C$, $p_{\min}$, $MinTrue$)
18:     $C_{new} \leftarrow \emptyset$
19:     for all $p \in C$ do
20:         if $p_{\min} \preceq p$ then
21:             for $i \leftarrow 1, d$ do
22:                 $p' \leftarrow p; p'[i] \leftarrow p_{\min}[i] - 1/k$
23:                 if IsMaximal($p'$, $MinTrue$) then
24:                     $C_{new} \leftarrow C_{new} \cup \{p'\}$
25:                 end if
26:             end for
27:         else
28:             $C_{new} \leftarrow C_{new} \cup \{p\}$
29:         end if
30:     end for
31: end procedure
```

Algorithm 3 Finding a $MinTrue$ point

```
Require: $Pred$ is a monotone binary predicate
Require: $Pred(p) = true$
 1: procedure FindMinTrue($p$, $Pred$)
 2:     for $i \leftarrow 1, d$ do
 3:         $p \leftarrow$ FindMinTrueInDim($i$, $p$, $Pred$)
 4:     end for
 5:     return $p$
 6: end procedure

 7: procedure FindMinTrueInDim($i$, $p$, $Pred$)
 8:     $hi \leftarrow p[i] \cdot k$
 9:     $p[i] = 0$
10:     if $Pred(p) = true$ then
11:         return $p$
12:     end if
13:     $lo \leftarrow 0$
14:     while $hi - lo \geq 2$ do                    ▷ Binary search
15:         $mid \leftarrow (hi + lo)/2$
16:         $p[i] = mid/k$
17:         if $Pred(p) = true$ then
18:             $hi = mid$
19:         else
20:             $lo = mid$
21:         end if
22:     end while
23: end procedure
```

ple: once the algorithm finds a candidate with recall r, it focuses on regions of the similarity space with recall $> r$ and searches for candidates. To efficiently search for such candidates, we use the anti-monotonic property of recall stated earlier: for any two points $p_1 \preceq p_2$ implies $Recall(p_1) \geq Recall(p_2)$; equivalently, the predicate $Recall() \leq r$ is monotonic. If p_{mt} is a candidate with $Recall(p_{mt}) > r$, then there exists $MaxFalse$ point p of the predicate $Recall() \leq r$ such that $p_{mt} \preceq p$. Also, from monotonicity assumption, $Prec(p) \geq \tau$.

Based on the above observation, the algorithm simply considers points p in $MaxFalse(Recall() \leq r)$. If there exists p such that $Prec(p) \geq \tau$, it invokes FindMinTrue $(p, Prec() \geq \tau)$ to find a candidate with recall $r' > r$; this sets off a new iteration, and the algorithm searches for candidates with recall $> r'$. Naively checking for every point p in $MaxFalse(Recall() \leq r)$ if $Prec(p) \geq \tau$ by invoking the precision oracle might actually increase the number of precision oracle calls over AlgP. Algorithm AlgPR has additional logic involving $MaxFalse$ points of the predicate $Prec() \geq \tau$ to avoid this problem; we discuss these details in the full version of the paper.

4.2 Linear Classifier

We now discuss algorithms for learning a monotonic linear classifier with precision at least τ that maximizes recall. Recall that a monotonic linear classifier L is of the form $\sum_i w_i F_i \geq 1$. The linear classifier classifies a point $f = \langle f_1, \ldots, f_d\rangle$ as $true$ if $\sum_i w_i f_i \geq 1$, and $false$, otherwise. For monotonic linear classifiers w_i are real non-negative numbers.

We show that the problem of learning the best monotonic linear classifier can be reduced to the problem of learning the best conjunction of similarity thresholds, which we can solve using the algorithms of previous section.

We can define a natural partial-ordering over monotonic linear classifiers. Given two linear classifiers L_1 and L_2, we say that $L_1 \preceq L_2$ if $\forall f \in [0,1]^d$, $(L_2(f) = true) \Rightarrow (L_1(f) = true)$. Essentially, the set of points which L_2 evaluates to $true$ is a subset of those which L_1 evaluates to $true$. From monotonicity principle, it follows that if $L_1 \preceq L_2$ than $Prec(L_1) \leq Prec(L_2)$. Also, by definition, $Recall(L_1) \geq Recall(L_2)$.

The following theorem states that there exists a mapping from monotonic linear classifiers to points in k-granular points in d-dimensional space that preserves the above partial order.

Theorem 2. *There exists a partial onto mapping function $\mathcal{Q}$ from the class of monotonic linear classifiers to k-granularity points in $[0,1]^d$ such that:*

1. *For any two monotonic linear classifiers $L_1 \preceq L_2$, if both $\mathcal{Q}(L_1)$ and $\mathcal{Q}(L_2)$ are defined, $\mathcal{Q}(L_1) \preceq \mathcal{Q}(L_2)$; and*
2. *For any monotonic linear classifier L, there exists a monotonic linear classifier L' such that $\mathcal{Q}(L')$ is defined, and the volume of $|L - L'|$ is $\leq d/k$.*

Briefly, we run AlgP/AlgPR with a minor modification: in order to evaluate precision (resp. recall) of a point p, we first identify the linear classifier L corresponding to the point p, i.e., $\mathcal{Q}(L) = p$, and return the precision (resp. recall) of L, respectively. If p_{best} denotes the point returned by the algorithm, the output linear classifier is L_{best}, where $\mathcal{Q}(L_{best}) = p_{best}$. Details of the construction of $\mathcal{Q}$ and $\mathcal{Q}^{-1}$ are presented in the full version of the paper.

4.3 s-term DNF

An s-term DNF is of the form $\mathcal{M}_1 \vee \ldots \vee \mathcal{M}_s$ where each $\mathcal{M}_i$ is a conjunction of similarity thresholds. For example, $(Jaccard(Name) \geq 0.9) \vee ((Edit(Street) \geq 0.7) \wedge (Jaccard(City) \geq 0.6))$ is a 2-term DNF.

We now present a simple greedy algorithm for learning an s-term DNF $(\mathcal{M}_1 \vee \ldots \vee \mathcal{M}_s)$. The algorithm proceeds in s steps and the

Algorithm 4 ALGPR: Algorithm that exploits recall

```
1: $p_{best} \leftarrow null$                                  ▷ current best point
2: $r_{best} \leftarrow 0$
3: $MF \leftarrow \emptyset$
4: $foundNewPoint \leftarrow true$
5: while $foundNewPoint = true$ do
6:     $hiRecallPts \leftarrow$ ENUMERATEBOUNDARY($Recall \leq r_{best}$)
7:     $foundNewPoint = false$
8:     for all $p \in hiRecallPts$ do
9:         if ($\exists p_{mf} \in MF, p \preceq p_{mf}$) then
10:            continue
11:        end if
12:        if $Prec(p) \geq \tau$ then
13:            $p_{best} \leftarrow$FINDMINTRUE($p$, $Prec \geq \tau$)
14:            $r_{best} \leftarrow Recall(p_{best})$
15:            $foundNewPoint \leftarrow true$
16:            break
17:        else
18:            $p_{mf} \leftarrow$FINDMAXFALSE($p$, $Prec \geq \tau$)
19:            $MF \leftarrow MF \cup \{p_{mf}\}$
20:        end if
21:    end for
22: end while
23: output $p_{best}$
```

binary function $\mathcal{M}_i$ is learned in the ith step. Let $p^{(1)}, \ldots, p^{(i-1)}$ denote the points in $[0,1]^d$ corresponding to the functions $\mathcal{M}_1, \ldots, \mathcal{M}_{(i-1)}$, respectively, which were learned in the previous $(i-1)$ steps. For any point p, let $(p - p^{(1)} - \ldots - p^{(i-1)})$ denote the region of the similarity space containing all points p' such that $p' \succeq p$ and $\forall j (1 \leq j < i)\ p' \not\succeq p^{(j)}$. To learn $\mathcal{M}_i$, we run ALGPR of Section 4.1.2 with the following modification: instead of computing precision (resp. recall) of a point p, we compute the precision (resp. recall) of the region $(p - p^{(1)} - \ldots - p^{(i-1)})$. We can show that the resulting s-term DNF $(\mathcal{M}_1 \vee \ldots \vee \mathcal{M}_s)$ has precision $\geq \tau$. The algorithm does not have recall guarantees, however.

5. COMPUTING PRECISION AND RECALL

We now discuss the implementation of precision and recall oracles. The precision and recall oracles take as input a binary (classifier) function $\mathcal{M}$ and output the precision and recall of the function, respectively. The techniques that we propose do not compute precision values exactly, rather they estimate these values using sampling techniques. The estimated values are only probabilistically approximate, but this suffices for record matching applications.

Estimating the precision and recall values for a binary function $\mathcal{M}$ is generally hard for large input tables R and S, even for the limited class of functions that we consider. We essentially run into the same computational issues we face when trying to use $\mathcal{M}$ for record matching, i.e., identify all pairs of records $(r,s) \in R \times S$ such that $\mathcal{M}(r,s) = true$.

Our implementation of the precision and recall oracles exploits the existence of the blocking function $\mathcal{B}$ (see Section 2) in our problem formulation, i.e., we only seek record matching packages of the form $\mathcal{M} \wedge \mathcal{B}$. When the precision oracle (resp. recall oracle) gets a request for estimating the precision of a function $\mathcal{M}$, it simply returns an estimate for the precision (resp. recall) of the function $(\mathcal{M} \wedge \mathcal{B})$. In other words, our algorithms of Section 4 do not "know" about the blocking function $\mathcal{B}$, but we can show that with this modified implementation of the precision and recall oracles, with high probability the function $\mathcal{M}$ learned by our algorithms of Section 4 will satisfy $Prec(\mathcal{M} \wedge \mathcal{B}) \geq \tau \pm \epsilon$ (which approximates the specified threshold τ by ϵ) and the recall of $(\mathcal{M} \wedge \mathcal{B})$ will be maximum modulo this approximation.

We next discuss how to estimate the precision and recall of $(\mathcal{M} \wedge \mathcal{B})$ for an arbitrary binary function $\mathcal{M}$ and a blocking function $\mathcal{B}$. In a preprocessing step, we evaluate the blocking function $\mathcal{B}$ over R and S and materialize the set of all pairs $(r,s) \in R \times S$ such that $\mathcal{B}(r,s) = true$. By the definition of the blocking function, this evaluation is efficient, which also implies that the number of such pairs is relatively small. In the description below, we denote this set using $\mathcal{B}(R,S)$.

We use standard Monte-Carlo estimation techniques [16] to estimate the precision of $\mathcal{M} \wedge \mathcal{B}$. In particular, we identify a random sample of pairs $(r,s) \in R \times S$ that satisfy the predicate $\mathcal{M} \wedge \mathcal{B}$ and seek labels from the user for the pairs in the sample. The fraction of pairs labeled as a match is an estimate for the precision of $\mathcal{B} \wedge \mathcal{M}$. To identify a random sample of pairs that satisfy $\mathcal{M} \wedge \mathcal{B}$ we simply scan $\mathcal{B}(R,S)$, identify the subset of pairs that satisfy $\mathcal{M}$ and sample from this subset. We can reduce the number of samples required, and therefore the labeling effort by exploiting the fact, that the algorithms of Section 4 require precision only to check if it is above or below the threshold τ; we can use the algorithms proposed in [30] for this purpose. For computing recall of $(\mathcal{M} \wedge \mathcal{B})$, we simply scan $\mathcal{B}(R,S)$ and count the number of pairs that satisfy $\mathcal{M}$.

Reusing samples: We present a simple modification to the sampling step above (for estimating precision) that significantly reduces the number of labeled pairs in practice. We fix a random permutation π of all pairs in $R \times S$; in practice, this can be done using a random hash function over $R \times S$ and sorting the pairs by their hash values. To sample k points that satisfy the predicate $\mathcal{M} \wedge \mathcal{B}$, we pick from among all pairs $(r,s) \in R \times S$ that satisfy $\mathcal{M} \wedge \mathcal{B}$, the k smallest ones according to π. This modification preserves the probabilistic guarantees associated with precision estimation. Using a consistent ordering π of pairs in $R \times S$ increases the likelihood of an overlap in samples required for different precision estimations, which translates to fewer distinct label requests.

Passive Learning: The algorithms of Section 4 can also be used in a passive learning setting by using a different implementation of precision and recall oracles. In passive learning, the learning algorithm is provided a list of labeled example record pairs as input (and not tables R and S). The goal of the algorithm is to learn a record matching package that has precision $\geq \tau$ and maximum recall over the labeled examples. In this setting, the oracles evaluate the precision and recall of a function $\mathcal{M}$ over the input labeled examples. We do not require a blocking function for passive learning.

6. EXPERIMENTS

We now present our experimental results. We use our experiments to (1) illustrate the performance characteristics of our algorithms and highlight various design choices, (2) compare our algorithms with previously proposed algorithms for learning record matching packages using active learning [37, 40], and (3) illustrate the advantages of active learning over passive learning.

Datasets

We use two large real world datasets in our experiments. Each dataset contains two tables (R and S) over which record matching is performed. The first dataset, called ORG, contains two tables with a million organization records (e.g., see Figure 1) each. Figure 4 lists the columns of these records; for brevity, we abbreviate column names by their first letters. For record matching, we use 8 similarity dimensions listed in Figure 4. In Figure 4, $\mathcal{J}$, $\mathcal{JC}$, and $\mathcal{EQ}$ denote jaccard similarity, jaccard containment, and string

Size:	$\lvert R\rvert = 10^6$, $\lvert S\rvert = 10^6$ records
Columns:	*Name(N)*, *Street(S)*, *City(C)*, *Zip(Z)*, *Phone(P)*
Similarity:	$\mathcal{J}(N)$, $\mathcal{J}(S,C,Z)$, $\mathcal{J}(N,S,C,Z)$, $\mathcal{EQ}(P)$, $\mathcal{EQ}(C)$, $\mathcal{EQ}(Z)$, $\mathcal{JC}(N)$, $\mathcal{JC}(S,C,Z)$
Blocking Function:	$\mathcal{J}(N) \geq 0.7$, $\mathcal{J}(S,C,Z) \geq 0.7$, $\mathcal{J}(N,S,C,Z) \geq 0.7$, $\mathcal{EQ}(P) = 1$. (Union Size = 836815 record pairs)

Figure 4: Characteristics of ORG dataset

Size:	$\lvert R\rvert = 10^5$, $\lvert S\rvert = 10^6$ records
Columns(R):	*Citation(C)*
Columns(S):	*Title (T)*, *Authors (A)*, *Venue (V)*, *Year (Y)*, *Pages (P)*
Similarity:	$\mathcal{J}(C \leftrightarrow T,A,V,Y,P)$, $\mathcal{JC}(C \leftrightarrow T)$, $\mathcal{JC}(C \leftrightarrow A)$, $\mathcal{JC}(C \leftrightarrow V)$, $\mathcal{JC}(C \leftrightarrow Y)$, $\mathcal{JC}(C \leftrightarrow P)$
Blocking Function:	$\mathcal{J}(C \leftrightarrow T,A,V,Y,P) \geq 0.35$. (Size = 38216).

Figure 5: Characteristics of PUB dataset

equality, respectively; also, e.g., $\mathcal{J}(N)$ denotes jaccard over *Name* column, and $\mathcal{J}(S,C,Z)$ denotes jaccard similarity over concatenation of *Street*, *City*, and *Zip* columns. Figure 4 also defines the blocking function for this dataset: it is the disjunction of the four listed predicates. All these predicates involve jaccard or equality, which are known to be efficiently computable [20].

The second dataset, called PUB, is from publications domain. It has two tables R and S; table R contains 100000 unsegmented citation strings and S contains close to a million segmented publication records. Figure 5 lists columns, similarity dimensions, and the blocking function for this dataset. In Figure 5, we use $\mathcal{J}(C \leftrightarrow T,A,V,Y,P)$ to denote jaccard similarity between *Citation* column of R and the concatenation of all columns of S.

Algorithms

In our experiments, we use an implementation of ALGPR that exploits both precision and recall. We estimate precision and recall using sampling as described in Section 5. For estimating precision, we do not use theoretical sampling bounds, but heuristically estimate precision using a fixed number($= 20$) of samples.

We also implemented two state-of-the-art prior algorithms [37, 40]. As originally proposed, these algorithms do not scale to large input sizes, but as we observed in Section 1, we can easily extend these algorithms to exploit blocking. Both algorithms use a committee-based approach, and they differ primarily in how they form their committees: In [40], different classifiers learn from different subsets of data and in [37], randomness is introduced into the classification learning process. We denote these algorithms CMT-D and CMT-R, respectively. Both algorithms use decision trees for their classifiers. Our implementation of these algorithms uses the Weka machine learning library [26]. For our experiments, we instantiated these algorithms with the best choice of parameters suggested by the experiments in [37, 40]. For our experiments on passive learning, we use the classifiers provided by Weka [26].

Recall:	29526
Precision:	97%
Number of Labels:	84.2
Avg time per label:	0.69 sec
Package Complexity:	2.6 terms
Representative Package:	$\mathcal{B} \wedge (\mathcal{EQ}(Z) = 1 \wedge \mathcal{JC}(N) \geq 0.55 \wedge \mathcal{JC}(S,C,Z) \geq 0.05)$

Figure 6: Performance of ALGPR for ORG

Methodology

To evaluate a candidate algorithm, we measure the quality of the learned record matching package (precision and recall), the number of labels requested, and the time delay between two labeling requests. For interactive labeling, it is desirable that the time delay be small. We do not have golden truths for our datasets, so we estimate precision manually by labeling a sample of the output; throughout, we use a sample of size 40.

6.1 Performance of AlgPR

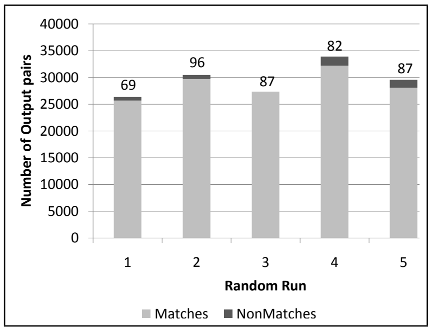
Figure 6 summarizes the performance of ALGPR for ORG dataset; the results are for learning a 1-term DNF. In other words, the the learned package is of the form $\mathcal{B} \wedge \mathcal{M}$, where $\mathcal{B}$ is the blocking function (Figure 4), and $\mathcal{M}$ is a conjunction of similarity thresholds. The statistics in Figure 6 are obtained by averaging over 5 runs with different random seeds, and we used $k = 20$ for granularity and $\tau = 0.95$ as precision threshold.

ALGPR learns a package with recall close to 30000, precision above 0.95 and requires about 84 labeled examples. The average time delay for each label request is under a second, suggesting that the algorithm is interactive. Figure 6 also shows a representative learned package. Figure 7(a) presents the results for each run separately. The number above each bar represents the number of labels requested. We note that the learned packages all have high precision, but their recall numbers vary by about 10%. This difference arises due to sampling estimation errors and should decrease if we increase the number of samples. The learned packages also differ, but their complexity is fairly small, typically 2-3 conjuncts. Although we present AlgPR in Section 4 with a single stopping criterion, we can easily modify it to stop at any point and output the current best package. Figure 8(a) plots the recall of three runs of ALGPR as a function of number of labeled examples. If we use ALGPR to learn 2-term DNFs, the recall of the learned package increases by about 2-5%, but the number of labeled examples required increases by about 50%. This suggests that the learning of the first term represents a point of diminishing returns.

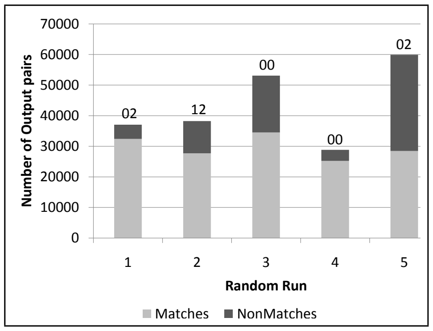
For the PUB dataset, ALGPR identifies a 1-term DNF with recall about 25000, precision close to 0.95 using 114 labeled examples. Most of the performance characteristics for PUB are similar to those for ORG dataset.

6.2 Comparison with Previous Algorithms

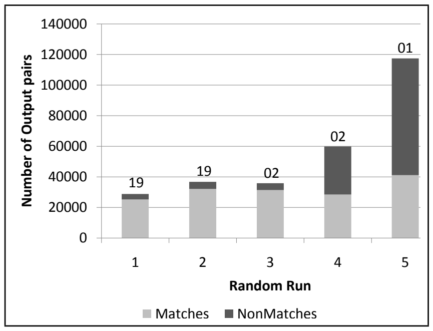
We now report on experiments comparing ALGPR with CMT-D and CMT-R. For the latter two algorithms, we started with a seed set of 5 matching pairs and 5 non-matching pairs picked randomly from a pool of labeled examples. Interestingly, both these algorithms behaved somewhat differently when run over candidate pairs picked using the blocking function, compared to running them over entire input data (as reported in [37, 40]). For both algorithms, all the classifiers in the committee reached *consensus* fairly quickly,



(a) ALGPR

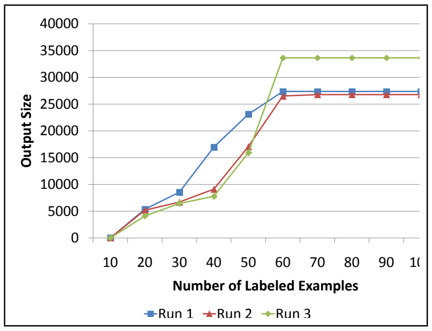


(b) CMT-R

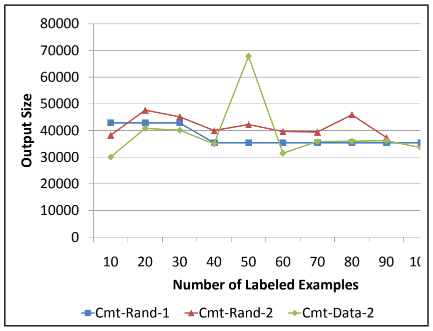


(c) CMT-D

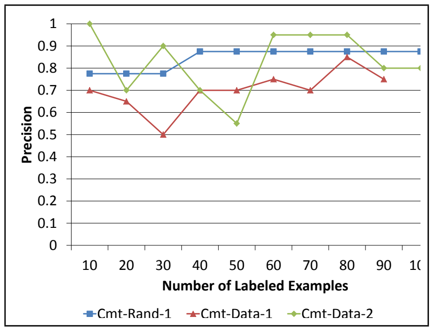
Figure 7: Performance of the algorithms for ORG dataset for 5 runs with different random seeds



(a) Recall of ALGPR



(b) Recall of CMT-D and CMT-R



(c) Precision of CMT-D and CMT-R

Figure 8: Evolution of learned package quality as a function of number of labeled examples

meaning that they classified all candidate pairs identically. We believe that the difference in behavior arises since the blocking function eliminates most of the "outliers" that cause disagreements between classifiers [37].

Figures 7(b) and 7(c) show the performance of CMT-R and CMT-D, respectively, on ORG dataset when we stop the algorithms when the committee reaches a consensus. Again, the numbers above the bars indicate the number of labeled examples used by the algorithms. We note that CMT-R and CMT-D sometimes reach consensus right at the beginning and therefore do not exploit active learning. Also, the performance of the algorithms varies widely from one run to another: the precision values of the learned packages range from 0.9 to less than 0.5 (run 5 in Figures 7(b) and 7(c)).

We also studied the alternative where we continue running the algorithms after the committee reaches a consensus, by labeling a randomly selected candidate pair, and adding it to set of labeled examples. We observed that, occasionally, the new example takes the committee out of consensus. Figures 8(b) and 8(c) present the recall and precision values (for ORG dataset) of two runs of CMT-R and one run of CMT-D as a function of number of labeled examples, using the above approach. The results of Figures 8(b) and 8(c) illustrate that the performance of two algorithms can be unpredictable and the quality of the learned classifier does not necessarily converge as the algorithms consume more examples. For example, in the run CMT-R-1, a high precision package is learned after 10 labeled examples; however, the precision drops to around 0.5 after 50 labeled examples. We also observed that the complexity of the learned package increases if it is not produced by an early consensus. For example, the decision tree learned by the run CMT-R-1 after 100 labeled examples contains 25 nodes.

Overall, these experiments highlight the difficulty of using CMT-R and CMT-D to learn a record matching package with high precision.

	ALGPR	C4.5
PRED1	(29311, 0.85)	(41767, 0.75)
PRED2	(36196, 0.85)	(43606, 0.60)

Figure 9: Precision recall values for passive learning

6.3 Passive Learning

Our final set of experiments highlight the advantages active learning over passive learning. As suggested by [8, 11], we study the approach of applying some (computable) predicate over the two input tables to identify a relatively small set of candidate record pairs. We then pick a sample from the candidate set, label the pairs in the sample, and use them as input to a learning algorithm.

For our experiments, we use two representative predicates over ORG dataset that roughly contain the same number of matches and non-matches (estimated). The first predicate (PRED1) is $\mathcal{EQ}(P) = 1$ (equal phone numbers), and the second (PRED2), $(\mathcal{J}(N) + \mathcal{J}(S, C, Z))/2 \in [0.48, 0.52)$. We picked the second predicate as Figure 2(c) suggests that the region of the similarity space where this predicate evaluates to true represents a transition from mostly matches to mostly non-matches. We labeled a random sample of size 100 from among the pairs that satisfy each of these predicates and use them as input to the C4.5 decision tree learning algorithm and ALGPR (with $k = 20$ and $\tau = 0.95$). Recall that ALGPR can be used in the passive learning setting by suitably modifying the precision and recall oracles.

Figure 9 summarizes the quality of the record matching package learned by the two algorithms, where the quality is measured over the entire ORG dataset. We note that the precision values of both algorithms are low. In particular, the precision value of the package learned by ALGPR (for either predicate) is only 0.85, although its precision over just the input labeled examples is close to 0.95.

7. RELATED WORK

Record matching is a well-studied problem and we refer the reader to [20] for a comprehensive survey of the topic. Previous work on record matching can be broadly divided into three buckets depending on how they relate to this paper: (1) identifying useful similarity functions (2) learning a record matching package, and (3) designing efficient string similarity join and lookup algorithms.

A variety of similarity (and distance) functions have been proposed for record matching [20]. These include domain independent functions such as edit distance, jaccard measure, and cosine similarity [14] and domain specific ones such as Jaro distance [29] for people names. Recall that similarity functions applied to attributes of input tables serve as similarity dimensions for our active learning problem. Chandel *et al* [10] present a comparison of different similarity functions for record matching.

A second line of work in record matching concerns with the learning of a matching package. Some of the earliest work in this area uses an *unsupervised* approach based on the *expectation maximization* algorithm [43]. Most of the recent work has focused on *supervised* approaches where a matching package is learned from labeled example record pairs. The supervised approaches can be passive, where the learning algorithm has no control over the labeled examples, or active, where the algorithm picks the examples to be labeled from among a large pool of unlabeled data. Section 1 presents a discussion of previously proposed supervised learning algorithms for record matching and how they relate to this paper.

A third line of work concerns with efficiency issues in record matching, and includes work on string similarity joins and lookups. Here the goal is to design efficient algorithms that identify pairs of records that have high similarity on some attribute. These algorithms are usually specific to a similarity function such as edit distance [23, 31] or jaccard [2, 25, 38]; Chaudhuri *et al* [12] propose a primitive based on set-similarity joins that can be used to implement string similarity joins for several different similarity functions. Many authors distinguish between simple equality based blocking (e.g., first letter scheme of Section 1) and more sophisticated schemes such as sorted neighborhood approach [28] and canopies [34]. Our definition is fairly general and subsumes the latter two. Bilenko *et al* [6] study the problem of learning a blocking function using labeled examples. Recent work [19, 39] has explored a "collective" approach to record matching that goes beyond pairwise similarity. A complete understanding of how this approach combines with traditional approaches is an interesting avenue of future work.

Active learning approaches are fairly well-studied in machine learning [15, 21] and have been applied to a variety of domains such as text classification [41] and information extraction [3]. As mentioned earlier, previous work on using active learning for record matching uses a classifier independent, committee-based approach [3]; classifier specific approaches have been proposed for SVMs [41] and decision trees [44]. There is also a large body of work in learning theory that explores the power of active learning compared to passive learning [17, 27]. Most of this work is in the *PAC* model (based on the misclassification rate quality measure; see Section 2) and not directly applicable to record matching for reasons mentioned in Section 2.

8. CONCLUSION

We presented new active learning algorithms for the record matching problem. Generalizing the techniques for finding maximal frequent itemsets, our algorithms cleverly navigate through the similarity space to find a conjunction of similarity thresholds. We are able to give probabilistic guarantees on the quality of the result while requiring fewer samples than passive learning algorithms. In an extensive experimental study we demonstrate the scalability of our approach to $10^6 \times 10^6$ record pairs with interactive response time. Compared to previous work on active learning our algorithms produce high-quality results more predictably using only around 100 labeled examples.

9. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. of the 28th Intl. Conf. on Very Large Data Bases*, pages 586–597, Aug. 2002.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proc. of the 32nd Intl. Conf. on Very Large Data Bases*, pages 918–929, 2006.

[3] S. Argamon-Engelson and I. Dagan. Committee-based sample selection for probabilistic classifiers. *J. Artif. Intell. Res. (JAIR)*, 11:335–360, 1999.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[5] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proc. of the 5th IEEE Intl. Conf. on Data Mining*, pages 58–65, 2005.

[6] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Proc. of the 6th IEEE Intl. Conf. on Data Mining*, pages 87–96, Dec. 2006.

[7] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 39–48, Aug. 2003.

[8] M. Bilenko and R. J. Mooney. On evaluation and training-set construction for duplicate detection. In *Proc. of the ACM SIGKDD-03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, Aug. 2003.

[9] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, June 1998.

[10] A. Chandel, O. Hassanzadeh, N. Koudas, et al. Benchmarking declarative approximate selection predicates. In *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, pages 353–364, June 2007.

[11] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *Proc. of the 33rd Intl. Conf. on Very Large Data Bases*, pages 327–338, 2007.

[12] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proc. of the 22nd Intl. Conf. on Data Engineering*, Apr. 2006.

[13] Citeseer. `http://citeseerx.ist.psu.edu/`.

[14] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Trans. on Information Systems*, 18(3):288–321, July 2000.

[15] D. A. Cohn, L. E. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[16] P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal of Computing*, 29(5):1484–1496, 2000.

[17] S. Dasgupta. Coarse sample complexity bounds for active learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 235–242. MIT Press, Cambridge, MA, 2006.

[18] The DBLP computer science bibliography. `http://www.informatik.uni-trier.de/~ley/db/`.

[19] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data*, pages 85–96, June 2005.

[20] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowledge and Data Engg.*, 19(1):1–16, Jan. 2007.

[21] Y. Freund, H. S. Seung, E. Shamir, et al. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[22] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *Proc. of the 3rd Intl. Conf. on Digital Libraries*, pages 89–98, June 1998.

[23] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, et al. Approximate string joins in a database (almost) for free. In *Proc. of the 27th Intl. Conf. on Very Large Data Bases*, pages 491–500, Sept. 2001.

[24] D. Gunopulos, R. Khardon, H. Mannila, et al. Discovering all most specific sentences. *ACM Trans. on Database Systems*, 28(2):140–174, June 2003.

[25] M. Hadjieleftheriou, A. Chandel, N. Koudas, et al. Fast indexes and algorithms for set similarity selection queries. In *Proc. of the 24th Intl. Conf. on Data Engineering*, pages 267–276, Apr. 2008.

[26] M. Hall, E. Frank, G. Holmes, et al. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[27] S. Hanneke. A bound on the label complexity of agnostic active learning. In *Proc. of the 24th Intl. Conf. on Machine Learning*, pages 353–360, 2007.

[28] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of Data*, pages 127–138, May 1995.

[29] M. A. Jaro. Unimatch: A record linkage system: User's manual. Technical report, US Bureau of the Census, Washington DC, 1976.

[30] R. M. Karp and R. Kleinberg. Noisy binary search and its applications. In *Proc. of the 8th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 881–890, Jan. 2007.

[31] C. Li, B. Wang, and X. Yang. Vgram: Improving performance of approximate queries on string collections using variable-length grams. In *Proc. of the 33rd Intl. Conf. on Very Large Data Bases*, pages 303–314, Sept. 2007.

[32] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of the 6th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 169–178, Aug. 2000.

[33] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[34] A. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of the 1st SIGMOD workshop on data mining and knowledge discovery*, 1997.

[35] G. N. Norén, R. Orre, and A. Bate. A hit-miss model for duplicate detection in the who drug safety database. In *Proc. of the 11th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 459–468, Aug. 2005.

[36] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.

[37] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 269–278, July 2002.

[38] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, pages 743–754, June 2004.

[39] P. Singla and P. Domingos. Multi-relational record linkage. In *Proc. of the 3rd KDD Workshop on Multi-Relational Data Mining*, pages 31–48, 2004.

[40] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, Dec. 2001.

[41] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

[42] W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington DC, 1999.

[43] W. E. Winkler. Improved decision rules in the Felligi-Sunter model of record linkage. Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington DC, 1993.

[44] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proc. of the 7th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 204–213, 2001.