Active Learning for Large-Scale Entity Resolution

Kun Qian
IBM Research – Almaden
qian.kun@ibm.com

Lucian Popa
IBM Research – Almaden
lpopa@us.ibm.com

Prithviraj Sen
IBM Research – Almaden
senp@us.ibm.com

ABSTRACT

Entity resolution (ER) is the task of identifying different representations of the same real-world object across datasets. Designing and tuning ER algorithms is an error-prone, labor-intensive process, which can significantly benefit from data-driven, automated learning methods. Our focus is on "big data" scenarios where the primary challenges include 1) identifying, out of a potentially massive set, a small subset of informative examples to be labeled by the user, 2) using the labeled examples to efficiently learn ER algorithms that achieve both high precision and high recall, and 3) executing the learned algorithm to determine duplicates at scale. Recent work on learning ER algorithms has employed active learning to partially address the above challenges by aiming to learn ER rules in the form of conjunctions of matching predicates, under precision guarantees. While successful in learning a single rule, prior work has been less successful in learning multiple rules that are sufficiently different from each other, thus missing opportunities for improving recall. In this paper, we introduce an active learning system that learns, at scale, multiple rules each having significant coverage of the space of duplicates, thus leading to high recall, in addition to high-precision. We show the superiority of our system on real-world ER scenarios of sizes up to tens of millions of records, over state-of-the-art active learning methods that learn either rules or committees of statistical classifiers for ER, and even over sophisticated methods based on first-order probabilistic models.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; **Data cleaning**;

KEYWORDS

Entity Resolution, Large-Scale Data Cleansing

1 INTRODUCTION

Entity Resolution (ER) identifies and links different representations of the same real-world entities. ER, also known as entity matching, record linkage, reference reconciliation, merge-purge (see [12, 14, 20, 26]), has become a central task in many of today's data analytics applications, since it is used to produce a unified, consistent view that subsequently enables other downstream entity-centric analysis. Despite receiving considerable attention from both the research community and industry, ER often requires extensive exploration and understanding of the data to formulate the matching algorithms. In both unstructured and semi-structured data scenarios, one needs to know which attributes are present and how frequently populated, which of these "identify" an entity or are relevant for matching, which are good comparison functions, and so on. Furthermore, the scale in modern "big data" applications adds to the challenge. As an example, assume that the ER task is to identify all matches between a set of social network users (e.g., from Twitter) and a set of Customer records (e.g., from Walmart). If each set contains 100 million (10^8) records, then the space of potential matches (the cross product) contains $10^8 \times 10^8$ pairs of records, and becomes prohibitive.

ER has a strong history of applying supervised learning approaches to learn a model that can then be used to detect duplicates [17]. The availability of labeled training data is a pre-condition for applying supervised learning. For ER, this usually implies labeling a large number of examples since the non-matches far outnumber matches and a substantial amount of human effort needs to be invested up front so that the training data consists of more than just a handful of matches. Going back to the Social-Customer scenario, let us further assume that the ratio of matches to non-matches is 10^{-3} on average[1]. Randomly exploring the space of examples, one would expect to label in excess of 10^5 pairs of records to obtain a training set with 100 matches. For this reason, recent work in ER has adopted active learning as a more guided approach to select the examples to label [2, 10, 21, 31, 34].

The main idea of active learning for ER is to reduce the number of pairs that need to be labeled by actively selecting the most informative examples. In [10, 21, 31, 34], the user is asked to label pairs of records that lead to maximum disagreement among a committee of models. One shortcoming of these early approaches is that they do not provide any quality control over the resulting models. Combined with the fact that they use randomization, such approaches lead to unstable results, where multiple runs over the same input can produce drastically varying precision. In contrast, Arasu et al. [2] use active learning to learn ER rules (i.e., conjunctions of matching predicates) whose precision exceeds a user-specified threshold. Arasu et al. ask the user to label a few examples covered by the current rule so that it can be refined into a more precise rule in case said examples turn out to be *false positives*. However, they ignore *false negative* examples, that is, examples of duplicates (or true matches) that are not covered by an existing rule. As a result, its recall suffers due to its inability to learn a second good rule or, more generally, a *set* of rules whose combined results may lead to significant coverage of the duplicates in the data.

In this paper, we introduce a new active learning system for ER that learns high-quality rules by searching for both false positives and false negatives. Our focus is on learning ER algorithms in

CIKM'17 , November 6–10, 2017, Singapore, Singapore
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132949

[1]Not an uncommon match rate in real-world, large-scale ER applications. Emp-Social in Section 4 exhibits a match rate $< 10^{-7}$.

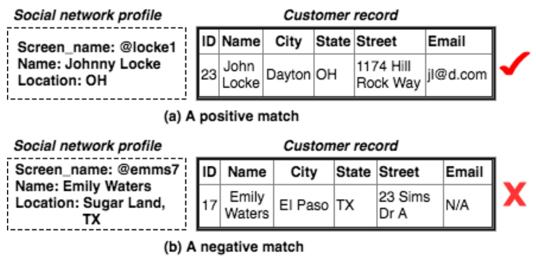
```
match Social S, Cust C by R1:
 S.lastname = C.lastname
 AND firstNameMatch(S.firstName,
                   C.firstName)
 AND S.location.state = C.state
                               ↪
```

(a)

```
match Social S, Cust C by R1′:
 lastNameFreqFilter(C.lastName,80)
 AND S.lastname = C.lastname
 AND firstNameMatch(S.firstName,
                   C.firstName)
 AND S.location.state = C.state
```

(b)

Figure 1: False positives can be used to refine rules.


Social network profile

Screen_name: @locke1
Name: Johnny Locke
Location: OH

Customer record

ID	Name	City	State	Street	Email
23	John Locke	Dayton	OH	1174 Hill Rock Way	jl@d.com

✓

(a) A positive match

Social network profile

Screen_name: @emms7
Name: Emily Waters
Location: Sugar Land, TX

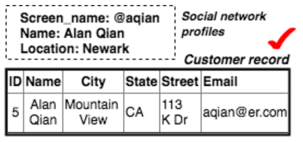
Customer record

ID	Name	City	State	Street	Email
17	Emily Waters	El Paso	TX	23 Sims Dr A	N/A

✗

(b) A negative match

Figure 2: Examples of matches: (a) positive, (b) negative.


Screen_name: @aqian
Name: Alan Qian
Location: Newark

Social network profiles

✓

Customer record

ID	Name	City	State	Street	Email
5	Alan Qian	Mountain View	CA	113 K Dr	aqian@er.com

(a)

```
match Social S, Cust C by R2:
 firstNameMatch(S.firstName,
               C.firstName)
 AND S.lastname = C.lastname
 AND sameHandle(S.screen_name,
               C.emailHandle)
```

(b)

Figure 3: False negatives lead to new rules.

big data scenarios, where the data may contain diverse attributes varying in sparsity (i.e., some may be present in some records, while others appear in other records). In such situations, learning just one rule may not achieve an acceptable level of recall; it is often advantageous, if not necessary, to learn multiple high-precision rules, each exploiting a different combination of attributes present in the data. Our primary hypothesis is that, while false positives can help refine a rule, false negatives are crucial for learning multiple rules. We illustrate next how the two kinds of examples fulfill different roles in the learning process.

Example 1.1. Figure 1(a) shows an initial rule R1 for our Social-Customer scenario, checking whether (1) the social network user profile and the customer record have the same last name, (2) the first names match via a user-defined function, and (3) the states are same. Figure 2 shows two examples satisfying R1, the second of which (Figure 2(b)) is marked as a false positive (non-duplicate) by the human labeler. As a result of this example, R1 can be automatically refined into the more conservative (and more precise) rule R1′ in Figure 1(b). This rule adds an extra condition specifying that it only applies to people whose last names are not in the top 80% frequent last names in the US. The refined rule has an improved precision and, in particular, avoids false positives such as the one in Figure 2(b). As opposed to false positives, false negative examples are matches that are not covered by any of the existing rules. Figure 3(a) shows a false negative where the name and email handles match perfectly (it is possible that Alan has moved since there is a Newark in California not far from Mountain View). More importantly, it represents a match that is not covered by R1′ (due to the missing state from the social network profile). With this example, our system may produce a new rule such as R2 in Figure 3(b) (in addition to R1′).

False negative examples enable the learning algorithm to form new combinations of matching conditions and exploit new attributes (e.g., email and social network screen handle, in the above example). While false positive examples are relatively easier to find, since they are part of the result of the current rules, finding false negatives is far more challenging. For our Social-Customer scenario, one high-precision rule may yield thousands of matches, while the search space of false negatives is as large as the remaining portion of the cross-product of the two datasets, i.e., 10^{16} pairs. To address this, we develop a novel and effective *Rule-Minus* heuristic, that actively searches for false negatives, by systematically removing one or more conditions from an existing rule.

The class of ER algorithms that we learn can be expressed as disjunctions of conjunctions of atomic predicates, where the predicates range from basic equalities on attributes to more complex, similarity or threshold-based comparison functions, and can also use filters and normalization functions. Due to the large scale aspect, part of the challenge is not only to be able to run the learned algorithms but, more importantly, the ability to learn the algorithms at scale. To this end, we build our learning system upon the HIL system [19] for large-scale entity resolution, which can compile ER rules into either Map/Reduce or Spark jobs to be executed using distributed infrastructure.

Besides utilizing active learning, a more recent approach to ER is to use crowdsourcing [11, 23, 36–39] by incentivizing humans to perform the matching task itself, often without learning an ER model. Such pay-as-you-go approaches make sense when the items being resolved are such that features are difficult to extract (e.g., resolving images). Note that, these techniques necessitate re-incentivization (in most cases, monetary) each time we have to perform ER. For instance in the Social-Customer scenario, following the first ER task of resolving 10^{16} pairs if we wanted to resolve a new set of social network users (since users are continuously being added) then this would need more crowdsourcing leading to significant costs. Our setting is quite different since we assume that the inputs are organized as relations whose tuples can be readily used to derive useful features for ER. In this case, learning an ER model makes more sense since it can be automatically reused to resolve new data as many times as needed and we find that post-hoc inspection of an interpretable model (e.g., rules) leads to more trust.

Our main contributions are summarized as follows:

- While previous work on active learning for ER either treats both types of examples equally [10, 21, 31, 34] or ignores one of the two [2], to the best of our knowledge, we are the first to identify the different roles fulfilled by false positives and false negatives during learning.
- We develop an overall active learning system to learn high quality entity resolution algorithms at scale on big data.
- As part of the system, we develop an algorithm that can learn a rule by maximizing recall while satisfying a high-precision constraint over a given set of labeled examples.
- Besides developing novel methods to actively search for false positives, we are the first to propose methods for finding false negatives leading to multiple non-trivial rules.
- We show experimentally that our approach outperforms state-of-the-art methods both in terms of quality of the learned algorithms and number of examples labeled by the user. Baselines

include sophisticated first-order probabilistic models based on Markov Logic Networks [30], ER rules learned via active learning [2], and active learning with committees of classifiers [31].

- We show that large-scale ER can be achieved by learning only a handful of short, human-readable rules. In our experiments, we detect duplicates in 2 large-scale real-world scenarios by using 3-7 learned rules, each consisting of roughly 4-5 easy-to-understand predicates.

2 PROBLEM DEFINITION

In this section, we formulate the entity resolution learning problem. We assume that the ER task in consideration has two input datasets, possibly the same, denoted by D_1 and D_2. We use $(r, s) \in D_1 \times D_2$ to denote a pair of records (in short, *a pair*) in the cross product of D_1 and D_2. An entity resolution algorithm A is a disjunction of rules $\mathsf{R}_1 \vee \ldots \vee \mathsf{R}_k$ where each R_i is a conjunction $P^i_1 \wedge \ldots \wedge P^i_{m_i}$ of matching predicates. Syntactically, we write an ER algorithm as follows:

```
match  D1 r,  D2 s  by
    R1:  $P^1_1(r,s)$ AND ... AND $P^1_{m_1}(r,s)$,
    ...
    Rk:  $P^k_1(r,s)$ AND ... AND $P^k_{m_k}(r,s)$
```

The following example illustrates the matching predicates that can appear in rules.

Example 2.1. For the Social-Customer scenario, consider the predicate `UpperCase(S.lastname)` = `UpperCase(C.lastname)` that tests equality of last names in a case-insensitive manner. This matching predicate composes a normalization function (`UpperCase`) with a projection function (`.lastname`) before applying the equality predicate. More powerful matching predicates may use longer (≥ 2) chains of function compositions. In addition to equality, one could use similarity functions (e.g., Jaccard, edit distance, or user-defined) that allow for variations in the attribute values. Besides matching predicates that operate on pairs of records, rules can include filters that apply on a single record. The earlier predicate `lastNameFreqFilter(C.lastname,`θ`)` is a filter that returns true if a last name is not in the top θ% of the US population in terms of last name frequency. As in previous work [2], we use fixed increments to discretize the continuous-valued parameters that may appear in filters or similarity functions. For instance, using increments of 10, the above θ takes values from the set $\{10, 20 \ldots 90\}$, leading to a set of concrete filters {`lastNameFreqFilter(C.lastname,10)`, ..., `lastNameFreqFilter(C.lastname,90)`}. The learning algorithm will learn which concrete filter to include in a rule (if any).

Given a set of pre-defined atomic functions (for equality, comparisons of strings, normalization, filtering, etc.), it is possible to systematically enumerate all the matching predicates relevant to the ER task at hand. Care must be taken to ensure syntactic validity (i.e., certain functions may only apply to certain attributes in the schemas, compositions of functions must type-check) and to avoid repeated application of the same normalization function within a matching predicate (which would lead to compositions of unbounded length). Let $O = \langle P_1, \ldots, P_N \rangle$ denote a fixed permutation of all matching predicates relevant for D_1, D_2. Then $F(r,s) = \langle P_1(r,s), P_2(r,s), \ldots, P_N(r,s) \rangle$, denotes the *feature vector* corresponding to the pair (r, s). Here, the value of the k^{th} feature in $F(r,s)$ is the evaluation of $P_k \in O$ on pair (r, s). Note that each feature is boolean (i.e., 0/1, or false/true). In this work, we generate feature vectors on demand since it is unrealistic to materialize all pairs or all feature vectors for large D_1, D_2.

Along the same line of thought, we avoid learning any rule R that may require enumerating $D_1 \times D_2$. We do so by ensuring that each rule R contains at least one *blocking predicate* which is a matching predicate that has the additional property of "logically" partitioning the input datasets into smaller blocks of records so that R only compares records within the same block [17]. Discovering blocking predicates is out of the scope of this work. We assume that blocking predicates in O have been pre-identified and our learning algorithm will only generate rules that include at least one of these. As an aside, previous approaches for learning blocking predicates [6, 27] can be used to initialize our approach.

For active learning of ER algorithms, we will need to estimate the quality of the learned rules. We use the standard metrics of precision and recall. We refer to pair (r, s) as a *link* of R if it satisfies all matching predicates in R. Denote by $links(\mathsf{R}) \subseteq D_1 \times D_2$ the set of all links resulting from applying rule R on D_1, D_2. If a human user identifies a pair (r, s) as a pair of duplicates then we say this pair is a *match*, otherwise it is a *non-match* (we assume the user has perfect knowledge). A *true positive* of R is a link of R that is a match, while a *false positive* is one that is a non-match. A *false negative* of R is a match that is not present in $links(\mathsf{R})$. Let $TP(\mathsf{R}), FP(\mathsf{R}), FN(\mathsf{R})$ denote the number of true positives, false positives, false negatives of R, respectively. Then precision and recall are defined as:

$$Prec(\mathsf{R}) = \frac{TP(\mathsf{R})}{TP(\mathsf{R}) + FP(\mathsf{R})},\ Recall(\mathsf{R}) = \frac{TP(\mathsf{R})}{TP(\mathsf{R}) + FN(\mathsf{R})}$$

The above definitions can be easily extended to an ER algorithm A, by taking $links(\mathsf{A}) = \cup_{\mathsf{R} \in \mathsf{A}} links(\mathsf{R})$.

Learning Entity Resolution Algorithms: Let $\tau \in [0, 1]$ be a precision threshold. The goal of ER learning is to determine an algorithm A that maximizes recall while each rule $R \in A$ has precision higher than τ:

$$\max Recall(\mathsf{A}) \text{ such that } Prec(\mathsf{R}) \geq \tau,\ \forall \mathsf{R} \in \mathsf{A}$$

2.1 Estimating Precision and Recall

The above notions of precision and recall are defined with respect to the actual datasets D_1 and D_2. An exact calculation of recall, for instance, would require labeling (as match or non-match) of all pairs in $D_1 \times D_2$, which is not feasible in practice. During active learning, we want to be able to gauge the precision and recall of the candidate rules, while asking the user to label only a small number of pairs. We refer to a pair (r, s) that is presented to the user for labeling as an *example*. An example is called *positive* if labeled as match by the user, and *negative* otherwise.

Estimating Precision. Given that a rule R may return thousands of links or more, especially in big data scenarios, it is infeasible to estimate precision by labeling all of $links(\mathsf{R})$. A common approach to circumventing this issue is to estimate precision on a randomly selected subset of links (see also [2]). We go a step further than this, and obtain a conservative estimate of $Prec(\mathsf{R})$ by carefully selecting a subset of links of R that have *low confidence*. Intuitively, this represents an "adversarial" set of examples; if R has high precision on this small adversarial set, then it is likely to have at least the same precision on a randomly selected set of links of R. We give more details in Section 3.

Estimating Relative Recall of Rules. We can show that the number of links can be a good estimate of the recall, as the precision threshold gets close to one. More concretely, rules with higher link count are estimated to have higher recall, given the precision constraint.

Proposition 2.2. *Assume that* $|links(\mathsf{R}_2)| < |links(\mathsf{R}_1)|$ *and* $Prec(\mathsf{R}_1) \geq \tau$. *Then:* $\frac{Recall(\mathsf{R}_2) - Recall(\mathsf{R}_1)}{Recall(\mathsf{R}_1)} < \frac{1-\tau}{\tau}$.

Proof. By the earlier definition, we have that:

$$Recall(\mathsf{R}) = \frac{TP(\mathsf{R})}{TP(\mathsf{R})+FN(\mathsf{R})} = \frac{TP(\mathsf{R})}{matches(D_1, D_2)},$$

where $matches(D_1, D_2)$ is the number of all possible matches over $D_1 \times D_2$. Since the denominator is a constant, to compare the recall of two rules we only need to compare the numerators ($TP(\mathsf{R})$). If we now assume that each rule R satisfies the precision constraint, we have that

$$\tau \leq Prec(\mathsf{R}) = \frac{TP(\mathsf{R})}{TP(\mathsf{R})+FP(\mathsf{R})} = \frac{TP(\mathsf{R})}{|links(\mathsf{R})|},$$

where $|links(\mathsf{R})|$ is the size of $links(\mathsf{R})$. This implies that:

$$\frac{TP(\mathsf{R})}{\tau} \geq |links(\mathsf{R})| \geq TP(\mathsf{R}) \geq \tau \cdot |links(\mathsf{R})|,$$

where the middle inequality follows from the fact that the true positives of R are a subset of $links(\mathsf{R})$. Thus, we can bound from above and below either $TP(\mathsf{R})$ or $|links(\mathsf{R})|$ using functions of the other, and thus $|links(\mathsf{R})|$ is a surrogate of $TP(\mathsf{R})$. In particular, we have that $TP(\mathsf{R}) \in [\tau \cdot |links(\mathsf{R})|, |links(\mathsf{R})|]$.

We can now bound the relative loss in recall, given that we estimate *Recall* via $|links|$ in our optimization problem, as follows. Assume that we have two rules R_1 and R_2, both satisfying the precision constraint. Then, we have:

$$\frac{TP(\mathsf{R}_1)}{\tau} \geq |links(\mathsf{R}_1)| \geq TP(\mathsf{R}_1) \geq \tau \cdot |links(\mathsf{R}_1)|$$
$$\frac{TP(\mathsf{R}_2)}{\tau} \geq |links(\mathsf{R}_2)| \geq TP(\mathsf{R}_2) \geq \tau \cdot |links(\mathsf{R}_2)|$$

Suppose that the ER learning task estimates that R_1 has better recall than R_2 because $|links(R_1)| > |links(R_2)|$. Moreover, assume that, in fact, R_2 has higher recall (i.e., $TP(\mathsf{R}_2) > TP(\mathsf{R}_1)$). The loss in recall (given that R_1 was chosen over R_2) is:

$$TP(\mathsf{R}_2) - TP(\mathsf{R}_1) \leq |links(\mathsf{R}_2)| - \tau \cdot |links(\mathsf{R}_1)| \leq$$
$$|links(\mathsf{R}_1)| - \tau \cdot |links(\mathsf{R}_1)| = |links(\mathsf{R}_1)|(1-\tau).$$

The relative loss on recall is then bounded as follows:

$$\frac{TP(\mathsf{R}_2) - TP(\mathsf{R}_1)}{TP(\mathsf{R}_1)} \leq \frac{|links(\mathsf{R}_1)| \cdot (1-\tau)}{\tau \cdot |links(\mathsf{R}_1)|} = \frac{1-\tau}{\tau}.$$

□

Proposition 2.2 implies that if the relative order of rules based on number of links is not in accordance with the order based on actual recall, then the relative loss of recall incurred is bounded by a monotonically decreasing function of τ. It is a consequence of the definitions of $Prec(\mathsf{R})$ and $Recall(\mathsf{R})$, and allows choosing rules based on $|links(\mathsf{R})|$ which is advantageous since it does not require a human labeler. Note that, for the above proposition, it is

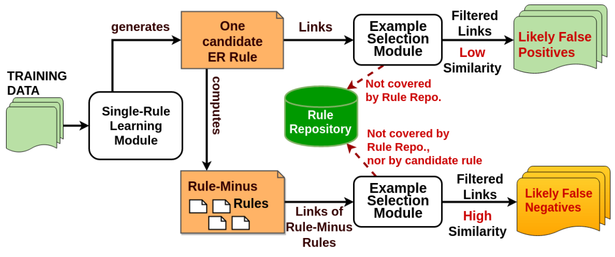


Figure 4: Automatic Rule Learning & Example Generation

important that the rules are learned under a precision constraint. In particular, the loss in recall is "small" when the precision threshold τ is "large" (e.g., $1/\tau - 1 \approx 0.05$ when $\tau = 0.95$). Moreover, we can control the degree of loss (increase τ for lower loss).

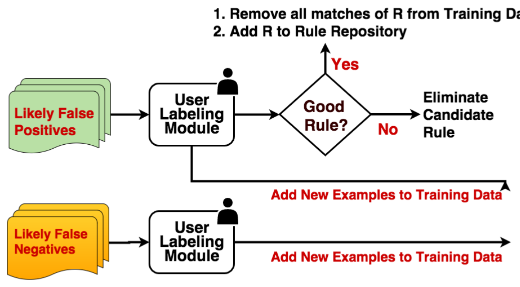
3 LEARNING SYSTEM

We describe our active learning system beginning with a high-level description of the general workflow (Figures 4 and 5). The major components include *Single-Rule Learning Module* (SRLM) which learns an ER rule from labeled examples, *Rule Repository* (RR) to store the learned ER rules, and *Example Selection Module* (ESM) which selects pairs to be labeled by the human user. Throughout the learning process, we maintain a training set T consisting of feature vectors for pairs along with their labels (match/non-match). Unlike other learning applications, T is not static and the system may add or remove examples to direct the learning of rules. Initially, T is a small set of pre-labeled examples perhaps obtained by some exploration of the space of matches (e.g., in Section 4, we use 30 initial examples). The learning process then proceeds in iterations each divided into two phases:

- The first phase (Figure 4) includes automatically learning a candidate rule from the current training set followed by choosing examples for the user to label.
- The second phase (Figures 5) includes the user interaction, where a user labels the chosen examples, followed by updating the training set with the newly provided labels. This phase also decides whether the candidate rule is accepted or rejected.

As earlier stated in the introduction, we differentiate between two kinds of examples: false positives (to refine rules) and false negatives (to improve the overall recall of the ER algorithm). Let R denote the candidate rule learned by SRLM in an iteration. To find false positives (upper branch in Figure 4), ESM chooses a few links from $links(\mathsf{R})$ that most resemble false positives for the user to label. We refer to these examples as *likely false positives*[2] or links with low confidence. Section 3.2 provides more details on this part of the workflow, including describing techniques to measure confidence of links. The candidate rule R will be accepted only if the user ratifies more than τ fraction of these as matches (i.e., if the vast majority of what the system thought as likely false positives turn out to be true matches). These requirements ensure that only high-precision rules are added to RR. This is also why, once a rule has been added, ESM never forwards a link from a subsequent

[2]Until the user labels them, we cannot be sure if they are indeed false positives hence the qualifier "likely".



1. Remove all matches of R from Training Data
2. Add R to Rule Repository
Yes
Likely False Positives
User Labeling Module
Good Rule?
No
Eliminate Candidate Rule
Add New Examples to Training Data
Likely False Negatives
User Labeling Module
Add New Examples to Training Data

Figure 5: User Interaction Phase

candidate rule R′ if it is already covered by an existing rule $R^\dagger$ in RR since $R^\dagger$, being high-precision, is unlikely to cover a false positive. Compared to likely false positives, computing likely false negatives (i.e., matches not covered by the current rules) is more challenging because non-matches tend to outnumber matches in ER. We propose the *Rule-Minus* heuristic, which uses rules derived from R to explore links that are not present in $links$(R) (lower branch in Figure 4). Since the goal is to find matches that will eventually lead to new rules that are sufficiently different from previously learned rules, ESM chooses high confidence links that are not covered by R or by rules in RR. Section 3.3 provides more details on how we compute likely false negatives.

Figure 5 shows the user interaction phase, in which the user labels the examples chosen by ESM which are then added to T as new labeled feature vectors. If a sufficient number of likely false positives were not labeled as matches then R is rejected, otherwise it is accepted. In the former case, with the newly labeled examples (of which we expect many were non-matches), we proceed to the next learning iteration where we hope to learn a more refined (more precise) ER rule. In the latter case, we include R into RR and remove its matches from T to ensure that the next iteration learns a rule that covers a new set of matches from T and hence will be sufficiently different from the previous rule. We continue iterating until either the user stops the system or the system can no longer find a new rule whose precision is above the threshold. The final ER algorithm is given by the disjunction of all rules in RR.

Example 3.1. (Continuation of Example 1.1) If R1′ (Figure 1(b)), produced as a refinement to R1 (Figure 1(a)), is high-precision then the system will accept it into RR, remove all its matches from T and proceed to the next iteration to learn rules such as R2 (Figure 3(b)). If the system is stopped at this point, then the learned ER algorithm is A = R1′ ∨ R2.

3.1 Single-Rule Learning Module

Algorithm LearnRule (Algorithm 1) learns a single ER rule containing a subset of the available matching predicates $O = \langle P_1, \ldots P_N \rangle$, given the available training data T. The algorithm is inspired by Angluin's algorithm [1] for learning monotone Boolean formulas, with a few important differences. While Angluin's algorithm requires an oracle for exact learning of the target formula, LearnRule uses labels in the training data instead, and can be seen as a greedy adaption that aims to cover the largest number of positive examples in T among all explored rules. Additionally, our algorithm ensures that the learned rule satisfies certain conditions such as including a

Algorithm 1: LearnRule(T, τ)

```
input: Set of labeled feature vectors $T$, precision threshold $\tau$
1  $T^+ \leftarrow \{\langle F, y\rangle | \langle F, y\rangle \in T, y = \text{match}\}$
2  best.R ← false; best.covg ← 0                      /* stores best rule */
3  for $\langle F, y\rangle \in T^+$ do                 /* iterate over each match $(r, s)$ */
4      $R \leftarrow$ true
5      for $i \in 1, \ldots N$ do                      /* select all true predicates */
6          if $F_i$ then $R \leftarrow R \wedge P_i$        /* where $F_i = P_i(r, s)$ */
7      $(R', \text{covg}) \leftarrow$ Generalize($R$, $T$, $\tau$)
8      if $covg > best.covg$ then
9          best.R ← $R'$; best.covg ← covg
10 return best.R
```

blocking predicate and, more importantly, satisfying the precision constraint with respect to T.

In the algorithm, T^+ denotes the set of entries in T that were labeled as a match. LearnRule then converts each such match into a rule R by selecting all features that are true for that match. Initially, R is a very specific rule that possibly satisfies only one match. We say a rule S is a *generalization* of R if $links$(R) ⊆ $links$(S). LearnRule then explores generalizations of R (via the procedure Generalize at Line 7) by removing predicates to improve its coverage of matches in T, possibly at the expense of decreased precision. In general, to maximize coverage of matches, an optimal algorithm would need to explore the power set of O. By employing pruning strategies within the Generalize procedure, LearnRule's worst case time complexity is reduced to having a quadratic dependence on the number of predicates in O instead of being exponential. One of the strategies we use is to find the predicate from R which leads to highest coverage of matches in T among all the subrules of R that drop a single predicate, and then continuing only with the resulting rule for further generalization. Additionally, we only consider rules that contain a blocking predicate and satisfy the precision constraint $Prec_T(\mathsf{R}) = |T^+ \cap links_T(\mathsf{R})| / |T \cap links_T(\mathsf{R})| \geq \tau$, where $links_T(\mathsf{R})$ denotes the subset of T covered by R. In particular, whenever we encounter a rule that does not satisfy the precision constraint (i.e., that has been generalized too much) we stop exploring any of its subrules. LearnRule returns the (generalized) rule with highest coverage of T^+ among all the explored rules.

3.2 Active Learning to Improve Precision

Since T is usually a small training set, $Prec_T(\mathsf{R}) \geq \tau$ may not necessarily imply $Prec(\mathsf{R}) \geq \tau$ on $D_1 \times D_2$, for a rule R learned by LearnRule. In other words, R may overfit on small T. Thus, as we learn a candidate rule R, it is very important to accurately estimate its *generalization performance* on data not used to learn it, i.e., on $D_1 \times D_2 \setminus T$. As mentioned earlier, prior work [2] has used a k-sized random sample $S \subseteq links(\mathsf{R})$ to estimate generalization performance of a rule R; however, this still allows for the possibility that $Prec(\mathsf{R}) \geq \tau$ on S but not on $D_1 \times D_2$, especially for small k. Instead, we propose using a *biased* sample with *low confidence* links of R (or likely false positives). We describe two approaches for choosing low confidence (or adversarial) links.

One way to estimate link confidence is via *bootstrap* [13], a powerful statistical technique that can assign confidence to almost any statistic (say, the label of pair). To apply bootstrap, construct K $|T|$-sized training sets $\{T^1, \ldots T^K\}$ by uniformly sampling at random from T *with replacement*, which may lead to repeated inclusion of a labeled pair from T into T^i. Each T^i can be used to produce a rule $\mathsf{R}^i = \textsc{LearnRule}(T^i, \tau)$. To estimate confidence of a pair (r, s) in $links(\mathsf{R})$, where R is the rule learned from T, we count how many of the R^i's yield the same pair: $\text{conf}_{BS}(r, s) = \sum_i \delta_{\mathsf{R}^i}(r, s)/K$, where $\delta_{\mathsf{R}^i}(r, s) = 1$ if (r, s) satisfies R^i and 0 otherwise. The k lowest confidence links form the biased sample of adversarial links whose human-assigned labels can be used to estimate $Prec(\mathsf{R})$.

While bootstrap is principled, it is also expensive requiring $O(K)$ rule evaluations per link. An alternate way, which we employ in our system, is to use a similarity measure that estimates confidence directly from the link (r, s) by measuring how well r matches s. This alternate way has the advantage that it applies equally well in finding likely false negatives (see later Section 3.3). We assume that all matching predicates in O measure goodness of match (to some degree) rather than its counter-positive. For instance, as opposed to `S.lastname = C.lastname`, `S.lastname ≠ C.lastname` does not measure goodness of match and our system does not allow the latter. A simple heuristic to estimate confidence of link (r, s) in such a setting is then to count the satisfying predicates: $\text{conf}_{\text{SIM}}(r, s) = \sum_{P \in O} \delta_P(r, s)/|O|$. In the user interaction phase (Figure 5), the system asks the user to label the k lowest confidence links. If $Prec(\mathsf{R}) \geq \tau$ on these low-confidence links, then R can be accepted. Otherwise, we proceed to the next iteration to learn an improved rule using the newly labeled examples added to the training set.

A caveat about the preceding discussion is that we never compute the confidence of a link that is covered by a high precision rule present in the Rule Repository since it is less likely to produce false positives. We also allow users to skip labeling examples due to lack of information and utilize early-out mechanisms to save labeling effort. One such situation is when most of the first $i < k$ examples are labeled non-matches; if adding these examples to T guarantees $Prec_T(\mathsf{R}) < \tau$, where R denotes the candidate rule, then we do not require labels for the remaining links. We note that bootstrap has been used for ER [29] to compute label uncertainties for every pair in the cross product $D_1 \times D_2$, as do other works on active learning of ER algorithms [10, 15, 21, 31, 34]. In general, it is unrealistic to enumerate $D_1 \times D_2$ in large-scale settings ($|D_1 \times D_2|$ for Emp-Social used in Section 4 exceeds 2.3×10^{13}). To prevent such complications, LearnRule always includes a blocking predicate.

3.3 Active Learning to Discover Unseen Matches

While false positive examples are relatively easier to find, since they are part of the result of an existing rule, finding false negatives (or, matches that are "unseen" by the existing rules) is far more challenging. To search for false negatives, one needs to explore the pairs of records that are *uncovered* (i.e., not produced) by the existing rules. In this section, we first provide an analysis to show that the search space of uncovered pairs is almost as large as the size of the cross-product of the two datasets. This analysis makes essential use of the fact that the rules we learn are expected to be very precise. We then develop a natural and effective *Rule-Minus* heuristic, to actively search for good candidates of false negatives, by systematically expanding from the search space of covered pairs.

Proposition 3.2. *If a set Φ of rules cover M matches and $Prec(\mathsf{R}) \geq \tau, \forall \mathsf{R} \in \Phi$, then $FP(\mathsf{R}) \leq M(\frac{1}{\tau} - 1)$, $\forall \mathsf{R} \in \Phi$.*

This follows from $FP(\mathsf{R}) = TP(\mathsf{R})/Prec(\mathsf{R}) - TP(\mathsf{R})$ and $TP(\mathsf{R}) \leq M$. Thus, if each rule's precision exceeds τ, we obtain:

$$| \cup_{\mathsf{R}} \; links(\mathsf{R})| \leq M + \sum_{\mathsf{R}} FP(\mathsf{R}) \leq M[1 + n(\frac{1}{\tau} - 1)],$$

where n denotes the number of rules, and where the maximum may be obtained when each rule's false positives form a disjoint set. Let $\sigma|D_1 \times D_2|$ denote the total number of matches; here, $\sigma \in [0, 1]$ is the ratio of matches to pairs. A lower bound on the number of pairs uncovered by Φ is then:

$$|D_1 \times D_2| - M[1 + n(1/\tau - 1)] \geq |D_1 \times D_2|[1 - \sigma\{1 + n(1/\tau - 1)\}]$$

using $M \leq \sigma|D_1 \times D_2|$. In practical matching scenarios, σ is very small (close to zero); at, the same time we desire τ to be close to 1, and also the number n of rules to be not too large. As an example, if we choose $\sigma = 10^{-3}, \tau = 0.9$, and $n = 10$, the preceding lower bound says that the number of uncovered pairs is 99.8% of the cross-product $D_1 \times D_2$. Note that in contrast to estimating generalization performance of a rule, sampling uniformly at random is not an option for discovering unseen matches because the fraction of space containing the remaining matches is too small:

$$\frac{\sigma|D_1 \times D_2| - M}{|D_1 \times D_2|[1 - \sigma\{1 + n(1/\tau - 1)\}]} \leq \frac{\sigma}{1 - \sigma\{1 + n(1/\tau - 1)\}}$$

Assuming the same values for σ, τ, and n as before, the above upper bound is about 10^{-3}.

A desirable way to discover unseen matches (or false negatives) without enumerating $D_1 \times D_2$ is to focus on regions that are likely to have a higher concentration of matches. Let us first extend the notion of a generalization of a rule. Given R and *jump size* $j \geq 1$ (a small number), let $\mathsf{R}^-(j)$ be a *Rule-Minus* rule obtained by dropping *at most* j predicates from R. While a rule can have many generalizations, we restrict our attention to ones containing blocking predicates; we denote by $RM(\mathsf{R}, j)$ the set of Rule-Minus rules that can be obtained in such way. Now consider $links(\mathsf{R}^-) \setminus \mathbf{L}$ where $\mathsf{R}^- \in RM(\mathsf{R}, j)$, R denotes the candidate rule, and $\mathbf{L}$ contains the links covered by R and the rules in the Rule Repository. We hypothesize that $links(\mathsf{R}^-) \setminus \mathbf{L}$ harbors a higher concentration of matches relative to other areas of $D_1 \times D_2 \setminus \mathbf{L}$. Thus, we propose the *Rule-Minus heuristic* which selects a small sample from $links(\mathsf{R}^-) \setminus \mathbf{L}$ for labeling. Denote by P the set of (at most j) predicates included in R but not in R^-. Since any $(r, s) \in links(\mathsf{R}^-) \setminus \mathbf{L}$ satisfies all predicates in R barring P, (r, s) is more likely to be a match than a pair chosen arbitrarily which may not satisfy even a single predicate in R. Intuitively, R^- is a minor perturbation of R and a link resulting from it is more likely to be a match than any pair from the much larger space $D_1 \times D_2$. Procedurally, we choose top-m *high confidence* links (using conf_{SIM}) from each Rule-Minus rule of the candidate rule R while ensuring that none of these is covered by R or any of the rules in the Rule Repository. These links are then sent to the user for labeling (see lower branch of Figure 5).

Besides combining the top-m links from each Rule-Minus rule, we could also union the links from all rules before selecting the top few. However, this risks dropping all links from a particular rule. By performing the union after choosing top-m we preserve links from every Rule-Minus rule thus exploring different parts of the search space. Jump size j controls the size of the region where we look for unseen matches. While increasing j increases the size of

	Size	Predicates	Ground Truth
DBLP-Scholar	2,6K×64K	124	Yes
Emp-Social	50M×470K	68	No
Crystal	1.3M×1.3M	158	No

Table 1: Statistics of datasets

the explored search space via more general Rule-Minus rules, it can also result in exploring regions of lower match concentration. In Section 4 we explore the effects of varying j on the effectiveness of the Rule-Minus heuristic.

4 EXPERIMENTAL EVALUATION

We now present an empirical evaluation of our approach on multiple real-world scenarios ranging from de-duplicating smaller datasets with only atomic attributes to resolving entities in large-scale semi-structured data with nested and list-valued attributes. We compare against multiple baselines, including prior active learning methods for ER [2, 31] along with supervised approaches based on sophisticated statistical models [30]. We show that ours is the only approach that can learn multiple high-precision ER rules which maximize recall while incurring the fewest human-labeled examples. We also show that it is possible to de-duplicate large-scale data by learning just a handful of human-readable rules.

Datasets. We use a variety of similarity functions, e.g., equality, edit distance, trigram and token-based Jaccard similarity, to experiment with three real-world datasets. The publicly available DBLP-Scholar dataset [25] consists of 2,616 and 64,263 bibliographic entries from DBLP and Google Scholar, respectively, where the task is to determine if two records refer to the same publication using attributes such as title, author list, venue and year. In contrast to DBLP-Scholar, the Emp-Social scenario poses a much larger ER task consisting of 467,761 employee records from a large enterprise and 50 million user profiles from a popular online social network. Besides including atomic attributes such as `gender` and `url` (e.g., a homepage when available), this scenario also includes nested and list-based attributes such as `EMAIL_ADDRS` (a list containing personal and work emails of an employee), `HOME` (a struct with address `city`, `state` and `country` of a person), `name` (a struct with `first`, `middle`, `last` and `full_name`) and `occupation_mentions` (a struct with `job_type`, `category` and `date`). In addition, we also use a pre-computed list of variations of first names (`firstNameVars`) to facilitate name matching. While the previous two scenarios match entities across datasets, the third scenario (called here Crystal) contains only one dataset with 1,334,766 company mentions containing many duplicates (collected from various real-world sources such as S&P Capital IQ), and including attributes such as company `name`, `industry`, `sub-industry` and `country`. Table 1 provides a summary of these scenarios including the number of matching predicates obtained after automatic application of functions to the relevant attributes. We note that the number of predicates is larger than in prior approaches, making the ER learning task more difficult.

Baselines. We compare our approach, ERLEARN, with several supervised and active learning baselines, covering major classes of learning-based ER approaches. We include support vector machines (SVM) [35], one of the most accurate statistical classifiers, and Markov logic networks (MLN) [30, 33], which have the added advantage of learning rules in first-order logic. Among active learning baselines, we include one that learns ER rules (ALGPR) [2] and one that learns statistical classifiers (ALIAS) [31]. More precisely, ALIAS learns a committee of decision trees by randomly choosing a feature to split on. The user has to label examples whose predictions lead to maximum disagreement among the committee.

Method	Rules	Labels	Recall	Precision	F-score
ALIAS	n/a	160	0.82±0.02	0.75±0.11	0.78±0.08
ALGPR	2	210	0.67	0.97	0.80
ERLEARN	6	163	0.84	0.90	0.87

Table 2: Active learning results on DBLP-Scholar

Methodology. To measure the efficacy of an ER algorithm A, we report precision and recall, as well as the number of labels incurred $|labels(\mathtt{A})|$ (i.e., the labor cost for learning A). To further understand the label-accuracy tradeoff, we define *cost effectiveness* as $\alpha(\mathtt{A}) = |links(\mathtt{A})|/|labels(\mathtt{A})|$, measuring the number of links produced by algorithm A per label incurred during learning. We give statistics for each individual rule and also look at the number of rules that are learned. Of the three datasets, two have no ground truth available. For such cases, we estimate precision by manually inspecting 30 randomly selected links, and estimate recall (see Proposition 2.2) via the total number of links produced at high precision. We also explore the effect of varying jump size (j) for Rule-Minus rules on ERLEARN's performance. We estimate link confidence using conf_{SIM} and unless otherwise stated, use the following parameter settings for ERLEARN: $\tau = 90\%$ (precision threshold), $k = 10$ (number of likely false positives produced per iteration) and $m = 2$ (number of likely false negatives per Rule-Minus rule).

Implementation. ERLEARN is implemented in Java, while the execution of the ER rules during active learning is done by passing the rules to the HIL system [19], which compiles and executes them on Spark[3]. The Spark cluster we used has 7 nodes (one master and six slaves), where each node is a 64-bit machine with 96GB of main memory, 24 cores and hyper-threading technology.

4.1 Comparison with Prior Work

Table 2 presents results of all three active learning methods on DBLP-Scholar using the same initial training set containing 5 matches and 5 non-matches. For these experiments, we set $j = 1$ and m=1 for ERLEARN. Also note that ALIAS is a randomized algorithm, so we report the average of 5 runs along with standard deviations. Of the three approaches, only ERLEARN and ALGPR adhere to the precision constraint ($\tau = 90\%$). The most noticeable aspect of ALIAS is its high standard deviation relative to its averages (also observed in [2]), especially in its precision. While more runs may reduce instability, each run incurs more labels resulting in increased human labor (the 5 runs in Table 2 incurred a cumulative cost of 800 labels). In contrast, ERLEARN significantly exceeds precision and F-score (harmonic mean of precision and recall) of ALIAS by learning 6 rules while incurring only 3 extra labels. While ALGPR achieves high precision, its recall is significantly lower than ERLEARN's and incurs almost 29% more labels. The next experiment provides a more challenging, large-scale setting where we also compare against supervised ER learning approaches.

[3]spark.apache.org

To train supervised models from Emp-Social, we use a training set consisting of 430 labeled pairs, collected via an active learning run of ERLEARN. We also use a blocking predicate (`social.lastname = Emp.lastname`) to avoid materializing the full cross product which is infeasible for Emp-Social. The initial training dataset used to start ERLEARN contains 30 labeled examples. Table 3 lists results of all methods on Emp-Social (bold font denotes links at high precision). While SVM and MLN do not adhere to a precision constraint, they both return a confidence score for each link. By choosing an appropriate threshold on the confidence score, it is possible to increase precision but at the expense of the number of links. Concretely, SVM returns only 21 links at high precision, while the more sophisticated MLN fares only slightly better with 84 links. As in the case of DBLP-Scholar, ALIAS's precision fell short of our 90% threshold but we estimate it to produce more matches than SVM or MLN ($340 \times 0.4 = 136$). Our MLN rules were learned using structure learning algorithms that can learn relatively more complex and long-ranging rules [24]. However, the 3 learned rules were too imprecise to provide a significant number of reliable links which illustrates the difficulty of learning ER rules in complex, large-scale settings. ALGPR is the only baseline whose cost effectiveness is greater than 1 (implying that the number of links exceeds labeled examples provided) and produced 2 rules whose precision exceeds τ but is still easily outperformed by ERLEARN. We report ERLEARN results with two settings of j. At j=1, ERLEARN produced 4 high-precision rules resulting in 683 links while incurring almost the same number of labels as ALGPR (translates to 3.5× cost effectiveness) and at j=2, ERLEARN produced 7 rules adding 405 high-precision links although this comes at the cost of increased labeling effort (271 more than ALGPR's). At higher j, ERLEARN drops more predicates from the candidate rule, thus resulting in more general Rule-Minus rules exploring a larger space (see Section 3.2) but at a higher labeling cost. The two choices for j correspond to different points in the trade-off between ER effectiveness and labeling effort. Note that we do not have too many jump size settings to try since at $j = 2$, usually, the blocking predicate is the only predicate left from the candidate rule.

In addition to using HIL, which is a deterministic system for evaluating ER rules, we also used Probabilistic Soft Logic (PSL) [3], a probabilistic inference engine, to evaluate ER rules learned by ERLEARN with $j = 2$. PSL is a first-order statistical inference framework that has been demonstrated to produce high-quality ER results [3]. Note that PSL does not learn rules, instead, it learns weights for a given set of rules to possibly differentiate among them. We gave PSL the rules learned by ERLEARN as well as the 430 user labeled examples, and then used the learned model (weighted rules) to predict links using PSL's inference engine. We observed that PSL produced 1,090 links with precision 0.9, which in turn gives rise to a cost effectiveness of 2.534. Compared to the last row of Table 3, results of HIL and PSL are indistinguishable. This shows that ERLEARN's rules are robust enough to be used with deterministic or probabilistic evaluation engines. More importantly, given that statistical inference is relatively expensive (#P-complete in the most general case), this result shows that learning *precise* ER rules enables the use of more efficient deterministic systems (e.g., HIL) without loss of accuracy.

We also applied ERLEARN to the Crystal scenario. Concretely, ERLEARN produces 3 rules each satisfying the precision constraint resulting in approximately 145,000 links while incurring 147 labels. This illustrates how ERLEARN is up to the task when it comes to a diverse range of ER applications, whether it is cleaning a single dataset containing thousands of duplicates (e.g., Crystal) or determining the (relatively few) matches across datasets which are largely free of duplicates on their own (e.g., DBLP-Scholar, Emp-Social).

Method		Labels	Links	Precision	α
SVM		430	**21**	0.952	0.049
MLN		430	**84**	0.90	0.195
ALIAS		241	340	0.40	0.56
ALGPR		159	**181**	0.933	1.14
ERLEARN	$j = 1$	172	**683**	0.933	3.97
	$j = 2$	430	**1088**	0.933	2.533

Table 3: A comparison of all approaches on Emp-Social. Bold font denotes links at high precision.

4.2 Quality of Learned Rules and Other Aspects

Figure 6(a) shows two of the ERLEARN rules learned, one for Emp-Social and one for Crystal. Rule `R1` is easy to understand despite considering evidence from multiple attributes comparing first name (and variations thereof), last name (while considering its rarity in census data), and geographic locations. The ability to learn the right combination of predicates and parameters separates ERLEARN from other approaches. For instance, MLN learns a rule resembling `R1` but lacking the filter on last name frequency resulting in many false positives corresponding to two different people who have the same common last name. ALGPR also learns a version of `R1` but where the parameter to `lastNameFreqFilter` changes from 60% to 85%, thus leading to significantly lower recall. Similar observations apply to Crystal.

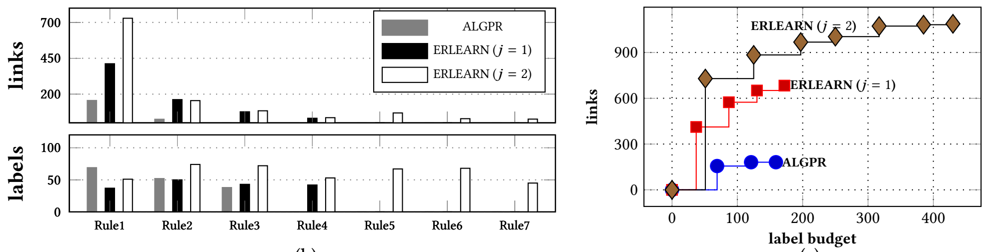
The upper chart of Figure 6(b) gives the count of *new* links that result from each rule learned by ERLEARN on Emp-Social, that is, links not covered by previously learned rules. The lower chart gives the number of labels incurred to learn each rule. We also provide similar numbers for ALGPR rules. The first rule learned by ERLEARN finds significantly more links, 412 with $j = 1$ and 729 with $j = 2$, than the two rules of ALGPR combined. While part of the credit belongs to ERLEARN for learning a very good first rule, this also points towards the ineffective subsequent rule learned by ALGPR[4]. ALGPR's second rule only returns 25 new links, which in comparison to its first rule (156 links) constitutes only 16% new links while incurring 52 labels (this is comparable to the 69 labels incurred to learn its first rule). In contrast, ERLEARN's subsequent rules provide significantly higher number of new links while incurring about the same (relatively low) number of labels (at $j = 1$, the 2nd, 3rd, 4th rule returns 162, 77, 32 new links, respectively). Thus ERLEARN's ability to learn more rules, each with higher recall, allows it to outperform other ER approaches. For our experiments we ran ERLEARN until it could not find a new rule that satisfied the precision constraint. In practice, the learning process can be stopped even earlier when the link contribution of a new rule goes below a certain threshold, or when the labeling budget has been exceeded. Figure 6(c) shows how ERLEARN performs when

[4]ALGPR's inability to learn subsequent rules is known [2].

```
match Emp i, Social s by R1:
 firstNameMatch(i.Name.firstNameVars,s.name.first)
 AND i.Name.last = s.name.last
 AND lastNameFreqFilter(s.name.last, 60)
 AND upperCase(i.CITY) = upperCase(s.home.city)
 AND countryIsInUSA(i.COUNTRY)

match Crystal c1, Crystal c2 by R2:
 c1.country = c2.country
 AND c1.industry = c2.industry
 AND c1.subIndustry = c2.subIndustry
 AND Jaccard_Similarity(c1.name, c2.name, 90)
```

(a)



(b) (c)

Figure 6: (a) Quality of rules learned by ERLEARN from Emp-Social (top) and Crystal (bottom). (b) Links obtained (top) and labels incurred (bottom) by each ERLEARN and ALGPR rule. (c) Number of reliable links obtained at various label budgets.

provided with a user-specified label budget B. It can be seen that ERLEARN produces (significantly) more links than ALGPR for all B.

Among other experiments, if we turn off the Rule-Minus heuristic, ERLEARN yields only one rule producing 162 links on Emp-Social, which is close to the ALGPR result (Table 3). To further analyze the efficacy of the Rule-Minus heuristic, we evaluated the ratio of the likely false negatives that are actual matches. On Emp-Social with $j = 1$ (2), out of 106 (238) likely false negatives, we found 82 (102) that were matches, representing a conversion rate of 78% (42.8%). Considering that Emp-Social has more than 2.3×10^{13} pairs, for Rule-Minus heuristic to be able to automatically discover these many matches (or any match at all) is remarkable. Some of these very matches may then lead to new high-precision rules that are learned by the LearnRule algorithm in subsequent iterations. These numbers point towards the Rule-Minus heuristic as the primary reason why ERLEARN does well on large-scale ER. In fact, it performs so well that when we increased the size of the initial training data from 30 to 80 examples, the results showed no discernible gain; thus, even with a small initial labeled dataset, the Rule-Minus heuristic makes up for it during the active learning phase.

Runtime efficiency. To evaluate the runtime performance in large-scale scenarios, we focus on the Emp-Social scenario. Each iteration of ERLEARN can be divided into three phases: single-rule learning, rule evaluation, and user interaction. Among the three, the rule evaluation and user interaction phases constitute more than 95% of the overall runtime. In terms of user interaction time, we found that on average ERLEARN incurs 6-7 labels per iteration with $j = 1$, and about 8-9 labels with $j = 2$. The time required to perform these labelings depends on the human user as well as on the scenario; in our experiments we spent one minute per label on average for the Emp-Social scenario, which leads to about 6-9 minutes of user time per iteration. In terms of rule evaluation, ERLEARN required about nine minutes per iteration on average when $j = 2$ in the Emp-Social scenario. This included the time to execute both the candidate rule and the Rule-Minus rules, via Spark jobs invoked by the HIL engine. One immediate optimization we applied to further improve the efficiency is to pre-compute the links (with features) that satisfy each blocking predicate and use the pre-computed links to speed up the evaluation of all candidate rules as well as the Rule-Minus rules. Concretely, for each blocking predicate p, we compute a set S of links that satisfy p, then for a given rule R that includes p, the evaluation of R can be done by scanning S to find out the links that satisfy the remaining predicates of R. With this optimization, we invest some time (8-9 minutes per blocking predicate) upfront to evaluate the blocking predicates, but we reduce the rule evaluation time from ~9 minutes to ~30 seconds per iteration.

5 RELATED WORK

One question that arises when comparing against supervised learning baselines is how to obtain the pre-requisite labeled training set. In our experiments, both SVM and MLN were afforded the luxury of readily available training data. While SVM[35] is one of the most widely used, large margin classifiers available, MLNs [24] represent a sophisticated statistical model that includes the very general first-order logic as a special case. In spite of this, both supervised learning baselines resulted in poor recall. Like ALIAS, other active learning ER methods also employ a committee [10, 21, 34]. While Tejada et al. [34] learn a committee of decision trees, de Freitas et al. [10] and Isele and Bizer [21] learn composite similarity functions using genetic programming. These approaches are based on the query-by-committee active learning strategy (QBC) [32] which can separate matches from non-matches perfectly if such a classifier exists [16]. In the more realistic non-separable case, QBC is not guaranteed to learn a good classifier [9] and as our experiments with ALIAS show, the QBC heuristic may produce unstable or poor results in comparison to ERLEARN. Fisher et al. [15] propose active learning with MLNs but harbor high expectations from the human user requiring a manually written rule whenever a pair is labeled incorrectly. In contrast, ERLEARN keeps the manual effort low by only requiring labels from the user.

Another line of work aims to ease supervised learning for ER by minimizing the manual effort required to select the subset of pairs to form labeled training data [7, 8]. However, the collected training data tends to be biased towards homogenous and easy-to-label pairs, and it is unclear how accurately the resulting classifier will label the full data set. Crowdsourcing is another approach for ER [29, 38] that uses similar techniques such as blocking [38] or a committee of classifiers [29] to reduce the number of pairs forwarded to the crowd of human labelers. The major focus however, is to distribute the labeling effort and ensure accurate labels despite different people labeling with varying accuracy. In contrast, for our experiments, one labeler (the first author) was enough to provide all the labels needed by ERLEARN even for large-scale scenarios.

Besides rules, one can learn a statistical classifier with precision guarantees [4] by repeatedly calling the IWAL active learner [5]. IWAL guarantees that the learned classifier's error rate will be equal to the classifier learned by supervised learning given n labels but may require $O(n)$ labels, in the worst case, to do so. Active learning theory suggests it may not be possible to do much better [22]. Instead of attempting to match the error rate obtained by supervised learning, ERLEARN aims for a different goal, that of maximizing the number of reliable links, and we empirically show how to achieve this while avoiding high label complexity.

The Emp-Social scenario bears a passing resemblance to user profile matching across social networks [28, 40]. Due to difficulties in obtaining entire social networks, user profile matching experiments are performed on subsets of the network whose characteristics are distinct and their ER results no longer apply to real-world social networks [18]. In contrast, we conduct experiments on real-world data and instead of subsampling, include the entire dataset.

6 FURTHER REMARKS AND CONCLUSION

We introduced a new system for scalable deduplication that learns high-quality ER algorithms using active learning. Our system is flexible enough to be able to use different methods for estimating link confidence and can also incorporate a label budget by stopping at any point and forming an ER algorithm with the rules learned thus far. To produce high-quality results, ERLEARN uses various techniques to discover false positives and false negatives, which enable the system to learn diverse rules each significantly adding to the count of reliable links (see Figure 6 (b)). While other active learning methods for ER may result in poor/unstable precision (ALIAS [31]) or low recall (ALGPR [2]), ERLEARN is the only method that achieves both high precision *and* high recall by learning only a handful of rules while keeping the labeling effort to a minimum.

Two reasons for ERLEARN's promising results include the discovery of unseen matches and ensuring precision of the learned rules. Traditionally, machine learning experiment design suggests use of a validation set to estimate generalization performance during learning. However, it may not be possible to estimate the generalization performance of every candidate rule produced using the same validation set V. For instance, let V contain a small number of labeled examples (say, 50). It is conceivable that rule R, which is precise on $D_1 \times D_2$, may be such that the intersection $V \cap links(\mathsf{R})$ is so small that it does not allow for accurately estimating R's generalization performance. Instead of using a static V, ERLEARN estimates precision by acquiring labels for a subset of $links(\mathsf{R})$ effectively *adapting* its validation set to each candidate rule. On the other hand, the proposed Rule-Minus heuristic successfully discovers unseen matches *without enumerating the cross product.* Rule-Minus heuristic is so effective that when we turned it off then the recall of ERLEARN, on Emp-Social, decreased significantly. Furthermore, the fact that ERLEARN successfully finds unseen matches on Emp-Social illustrates Rule-Minus heuristic's effectiveness even when matches are rare (when jump size is 1, there are 82 matches out of 2.3×10^{13} pairs, which constitutes 3.5×10^{-10}% of the cross product). In the future, we intend to evaluate ERLEARN on more datasets, devise ways to configure its parameters, and improve its convergence criteria.

REFERENCES

[1] D. Angluin. 1988. Queries and Concept Learning. *Machine Learning* (1988), 319–342.
[2] A. Arasu, M. Götz, and R. Kaushik. 2010. On Active Learning of Record Matching Packages. In *SIGMOD*. 783–794.
[3] S. Bach, M. Broecheler, B. Huang, and L. Getoor. 2015. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *CoRR* (2015). arXiv:abs/1505.04406
[4] K. Bellare, S. Iyengar, A. Parameswaran, and V. Rastogi. 2012. Active Sampling for Entity Matching. In *KDD*. 1131–1139.
[5] A. Beygelzimer, J. Langford, T. Zhang, and D. Hsu. 2010. Agnostic Active Learning Without Constraints. In *NIPS*. 199–207.
[6] M. Bilenko, B. Kamath, and R. Mooney. 2006. Adaptive blocking: Learning to scale up record linkage. In *Workshop on Information Integration on the Web*. 87–96.
[7] P. Christen, D. Vatsalan, and Q. Wang. 2015. Efficient Entity Resolution with Adaptive and Interactive Training Data Selection. In *ICDM*. 1550–4786.
[8] G. Dal Bianco, R. Galante, M. Gonsalves, S. Canuto, and C. Heuser. 2015. A Practical and Effective Sampling Selection Strategy for Large Scale Deduplication. *IEEE TKDE* (2015), 2305–2319.
[9] S. Dasgupta and D. Hsu. 2008. Hierarchical Sampling for Active Learning. In *ICML*. 208–215.
[10] J. de Freitas, G. Pappa, A. da Silva, M. Gonçalves, E. Moura, A. Veloso, A. Laender, and M. de Carvalho. 2010. Active Learning Genetic programming for record deduplication. In *IEEE Congress on Evolutionary Computation*. 1–8.
[11] G. Demartini, D. Difallah, and P. Cudre-Mauroux. 2013. Large-scale Linked Data Integration using Probabilistic Reasoning and Crowdsourcing. *VLDB Journal* (2013), 665–687.
[12] X. Dong, A. Halevy, and J. Madhavan. 2005. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*. 85–96.
[13] B. Efron and R. Tibshirani. 1993. *An Introduction to the Bootstrap*. Chapman & Hall.
[14] I. Fellegi and A. Sunter. 1969. A Theory for Record Linkage. *J. Amer. Statist. Assoc.* (1969), 1183–1210.
[15] J. Fisher, P. Christen, and Q. Wang. 2016. Active Learning Based Entity Resolution using Markov Logic. In *PAKDD*. 338–349.
[16] Y. Freund, H. Seung, E. Shamir, and N. Tishby. 1997. Selective sampling using the query by committee algorithm. *Machine Learning* (1997), 133–168.
[17] L. Getoor and A. Machanavajjhala. 2013. Entity Resolution for Big Data. In *KDD*.
[18] O. Goga, P. Loiseau, R. Sommer, R. Teixeira, and K. Gummadi. 2015. On the reliability of profile matching across large online social networks. In *KDD*. 1799–1808.
[19] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. 2013. HIL: A High-level Scripting Language for Entity Integration. In *EDBT*. 549–560.
[20] M. Hernández and S. Stolfo. 1995. The Merge/Purge Problem for Large Databases. In *SIGMOD*. 127–138.
[21] R. Isele and C. Bizer. 2013. Active Learning of Expressive Linkage Rules using Genetic Programming. *Web Semantics: Science, Services and Agents on the World Wide Web* (2013), 2–15.
[22] Matti Kääriäinen. 2006. *Active Learning in the Non-realizable Case*. 63–77.
[23] A. Khan and H. Garcia-Molina. 2016. Attribute-based Crowd Entity Resolution. In *CIKM*. 549–558.
[24] S. Kok and P. Domingos. 2010. Learning Markov logic networks using structural motifs. In *ICML*. 551–558.
[25] H. Köpcke and E. Rahm. 2008. Training selection for tuning entity matching. In *QDB/MUD*. 3–12.
[26] N. Koudas, S. Sarawagi, and D. Srivastava. 2006. Record Linkage: Similarity Measures and Algorithms. In *SIGMOD*. 802–803.
[27] M. Michelson and C. Knoblock. 2006. Learning Blocking Schemes for Record Linkage. In *AAAI*. 440–445.
[28] M. Motoyama and G. Varghese. 2009. I seek you: Searching and matching individuals in social networks. In *Workshop on Web Information and Data Management*. 67–75.
[29] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden. 2014. Scaling up crowd-sourcing to very large datasets: a case for active learning. In *VLDB*. 125–136.
[30] M. Richardson and P. Domingos. 2006. Markov logic networks. *Machine Learning Journal* (2006), 107–136.
[31] S. Sarawagi and A. Bhamidipaty. 2002. Interactive Deduplication Using Active Learning. In *KDD*. 269–278.
[32] H. Seung, M. Opper, and H. Sompolinsky. 1992. Query by committee. In *COLT*. 287–294.
[33] P. Singla and P. Domingos. 2006. Entity Resolution with Markov Logic. In *ICDM*. 572–582.
[34] S. Tejada, C. Knoblock, and S. Minton. 2001. Learning Object Identification Rules for Information Integration. *Information Systems* (2001), 607–633.
[35] V. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag.
[36] V. Verroios and H. Garcia-Molina. 2015. Entity Resolution with Crowd Errors. In *ICDE*. 219–230.
[37] N. Vesdapunt, K. Bellare, and N. Dalvi. 2014. Crowdsourcing algorithms for entity resolution. In *VLDB*. 1071–1082.
[38] J. Wang, T. Kraska, M. Franklin, and J. Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *PVLDB* (2012), 1483–1494.
[39] S. Whang, P. Lofgren, and H. Garcia-Molina. 2013. Question Selection for Crowd Entity Resolution. In *VLDB*. 349–360.
[40] G. You, S. Hwang, Z. Nie, and J. Wen. 2011. SocialSearch: Enhancing Entity Search with Social Network Matching. In *EDBT*. 515–519.