SystemER: A Human-in-the-loop System for Explainable 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. To achieve scalability and the desired level of quality, the typical ER pipeline includes multiple steps that may involve low-level coding and extensive human labor. We present `SystemER`, a tool for learning explainable ER models that reduces the human labor all throughout the stages of the ER pipeline. `SystemER` achieves explainability by learning rules that not only perform a given ER task but are human-comprehensible; this provides transparency into the learning process, and further enables verification and customization of the learned model by the domain experts. By leveraging a human in the loop and active learning, `SystemER` also ensures that a small number of labeled examples is sufficient to learn high-quality ER models. `SystemER` is a full-fledged tool that includes an easy to use interface, support for both flat files and semi-structured data, and scale-out capabilities by distributing computation via Apache Spark.

PVLDB Reference Format:
Kun Qian, Lucian Popa, and Prithviraj Sen. SystemER: A Human-in-the-loop System for Explainable Entity Resolution. *PVLDB*, 12(12): 1794-1797, 2019.
DOI: https://doi.org/10.14778/3352063.3352068

1. INTRODUCTION

Entity Resolution (ER), also known as entity matching, record linkage, reference reconciliation, and merge-purge, identifies and links different representations of the same real-world entities. As a data cleaning step, ER yields a unified and consistent view of data, thus serving as a crucial pre-processing step for downstream applications, including knowledge base creation, text mining, and social media analysis. More precisely stated, given two (possibly identical) entity databases D_1 and D_2, the goal of ER is to determine for each entity pair $r \in D_1, s \in D_2$ whether they represent the same real-world object. When D_1, D_2 are the same, the task is to identify duplicates within the same database.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352068

While both non-learning and learning based approaches for ER have been developed, the latter type of approaches provide more automation and have flourished recently. However, most existing learning-based ER tools and systems (e.g., Magellan[4] , dedupe[1], pydedupe[2], HUMO[2]) rely on learning statistical, black-box models that are difficult for humans to interpret. In contrast, consider learning for instance the ER rule shown in Figure 1, expressed in first-order logic, and aimed at deduplicating scientific publications:

```
DBLP.title = ACM.title
AND DBLP.year = ACM.year
AND jaccardSim(DBLP.authors, ACM.authors)> 0.1
AND jaccardSim(DBLP.venue, ACM.venue)> 0.1
                          → SamePaper(DBLP.id, ACM.id)
```

Figure 1: An ER rule for matching publications.

As opposed to learning a statistical model whose logic is encoded in its feature weights (e.g., for logistic regression), the above rule clearly states that two publications are deemed to represent the same real-world entity if their titles and years of publication are identical, and (token-based) Jaccard similarities of their author lists and publication venues exceed a certain threshold. A collection of one or more of such rules represents a human-comprehensible ER model. In general, *explainable* ER models have many advantages such as allowing for verification of its logic by domain experts and even enabling customization and modification so as to drive improvements in the presence of feedback.

Orthogonal to the issue of explainability, an end-to-end ER learning pipeline (Figure 2) includes multiple steps where non-trivial system-level support is needed to reduce the burden on the user:

(i) ***Data preparation and blocking***: ER tasks require access to a variety of forms of data, ranging from flat csv files to richly structured data where the attributes may themselves be arrays or structured fields. Furthermore, to unburden the computationally heavy learning step to follow, it is advantageous to reject the obvious non-matches early on - an operation known as *blocking.*

(ii) ***Training data creation***: Most learning-based systems require copious amounts of labeled data upfront, which is a significant investment in terms of human labor. A system that can learn accurate ER models with less labeled data can directly contribute towards reducing the start-up costs associated with ER, and further enable its applicability to new domains or datasets that may not have been seen before.

[1] `https://github.com/dedupeio/dedupe`
[2] `https://github.com/gpoulter/pydedupe`

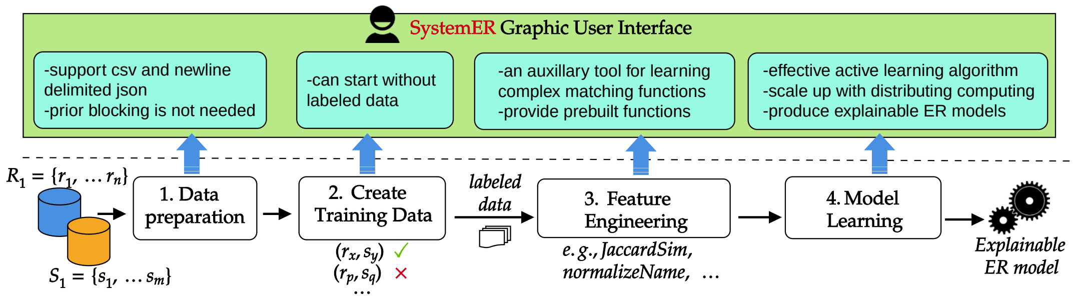

SystemER Graphic User Interface
-support csv and newline delimited json
-prior blocking is not needed
-can start without labeled data
-an auxillary tool for learning complex matching functions
-provide prebuilt functions
-effective active learning algorithm
-scale up with distributing computing
-produce explainable ER models
$R_1 = \{r_1, \dots r_n\}$
$S_1 = \{s_1, \dots s_m\}$
1. Data preparation
2. Create Training Data
(r_x, s_y) ✓
(r_p, s_q) ✗
...
labeled data
3. Feature Engineering
e.g., JaccardSim, normalizeName, ...
4. Model Learning
Explainable ER model

Figure 2: SystemER provides support for the multiple stages of the ER pipeline.

(iii) ***Feature engineering***: Identifying attributes and matching functions that contribute towards match/non-match decisions go a long way in achieving high quality ER. Usually, domain experts possessing in-depth knowledge of the application from which the data arises will be best-positioned to make such choices. However, domain experts may not possess low-level coding skills.

(iv) ***Scalable model learning***: Depending on the complexity of the learning algorithm and particularly on the complexity of the datasets, model learning itself may require distributing computation across a cluster.

We present SystemER, a tool that learns explainable ER models consisting of logical rules. A video demo is available at https://youtu.be/nsRAONFU_ak. SystemER is aimed at domain experts who are not necessarily programmers, but understand the data and the semantics of the task. SystemER minimizes the human effort as well as the required training data, by leveraging active learning techniques that choose only a few informative examples for the human to label while still learning a high-quality ER model. The high quality is achieved through a combination of multiple rules, which are individually above a precision threshold, and which collectively achieve high coverage (recall) over the space of possible matches. Every example chosen for labeling, by using SystemER's active learning algorithm (presented in [8]), is obtained by invoking some (intermediate) rule that is, in turn, interpretable, explainable and available for user's inspection if s/he desires. This provides explainability into the learning process. To best aid domain experts, SystemER also includes:

- an intuitive UI with support for end-to-end ER pipeline,
- prebuilt matching functions and an auxiliary tool, called LUSTRE[3], which learns sophisticated entity-specific normalization/matching functions for feature engineering,
- automatic support for blocking by including appropriate predicates into the learned rules,
- capabilities for running in single-node and cluster modes, via Spark[4], to scale up to large datasets.

Other related tools Among the existing systems, Febrl[5] and HUMO lack support for richly structured data, while additionally HUMO lacks support for blocking. Systems such as Magellan, dedupe, and pydedupe lack an easy-to-use UI to allow domain experts to construct complex matching functions without coding. JedAI[6] assumes that all data fits in main memory of a single machine and thus cannot scale to large inputs. Besides the systems mentioned above, [5] conducted a thorough study of 33 ER systems including both learning and non-learning based systems to identify shortcomings in each of them, thus underscoring the need for a system that provides end-to-end support for learning ER models. Finally, none of the above mentioned systems learn explainable models.

2. SYSTEM OVERVIEW

SystemER expects the user to understand the semantics of the task and have sufficient knowledge to correctly label selected entity pairs as matches or non-matches. SystemER has a carefully designed UI that hides the technical details and complexity of the ER pipeline from the user, while still makes the learning process highly explainable. We next elaborate how SystemER supports the ER pipeline and the user experiences with SystemER.

2.1 Data Preparation

As opposed to existing systems (e.g., Magellan and HUMO) that supports only flat csv files, the user can load csv files (assuming first row contains attribute headers) or newline delimited json files into SystemER, which can automatically infer the schemas of the input files. SystemER does not require any a priori blocking to be applied to the input data (as opposed to HUMO) nor does it require the user to write low-level code for blocking (as opposed to Magellan). Instead, the user just needs to provide the raw data files consisting of entity records. During feature engineering, the user is provided with an intuitive drag-and-drop UI to setup blocking.

Training data is optional. As opposed to [8], initial labeled data is optional in SystemER. If no labeled data is provided, SystemER has heuristics (see Section 2.3) to come up with a candidate rule for the first active learning iteration.

2.2 Feature Engineering

SystemER considers boolean features only, which is a design choice we made to facilitate the derivation of explainable ER models. This is not necessary a limitation since we can discretize a continuous feature (e.g., `jaccardSim(r,s)`) into multiple boolean features (e.g., `jaccardSim(r,s)>0.05`, ..., `jaccardSim(r,s)>0.95`). Additionally, all features in SystemER measure the *goodness* of match rather than its contrapositive. For example, as opposed to $\texttt{r.name} = \texttt{s.name}$, the predicate $\texttt{r.name} \neq \texttt{s.name}$ does not measure goodness of match and SystemER does not allow the latter.

SystemER provides three types of prebuilt functions: (1) *binary matching* (e.g., `jaccardSim(r.venue,s.venue)>0.2`), (2) *unary filtering* (e.g., `isUSA(r.country)`), and (3) *unary normalization* (e.g., `CompanyNorm(r.name)`). These functions include both general functions (e.g., `equality`) and

[3]Video demo of LUSTRE: https://youtu.be/hhM3BkvTZbE

[4]https://spark.apache.org/

[5]http://users.cecs.anu.edu.au/~Peter.Christen/Febrl/febrl-0.3/febrldoc-0.3/manual.html

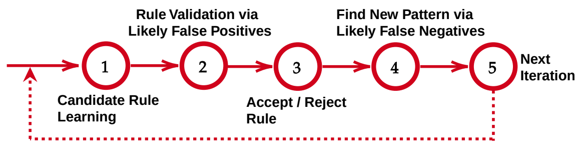


Figure 3: Workflow of one active learning iteration

entity-specific functions (e.g., `CompanyNorm`, `medicalTermSim`). Figure 4a shows `SystemER`'s UI for feature engineering. The user can click on a pre-built function to apply it and choose applicable attribute(s) by selecting from a dropdown list.

Extensibility. Although no programming skill is required, if the user has Java/Scala programming skills, s/he can add a new function by providing its Java/Scala implementation through the UI. `SystemER` also provides an auxiliary tool, named `LUSTRE` [1, 7], to learn entity-specific normalization/matching functions. For example, `LUSTRE` is able to learn a matching function that gives a similarity score of 1.0 for "`General Electric Corporation`" and "`GE Corp`" even if they are textually dissimilar.

Blocking. Users can use `SystemER`'s drag-and-drop UI (see Figure 4b) to setup "blocking buckets". Each bucket contains a conjunction of matching functions including at least one equality based function. During active learning, `SystemER` considers only candidate rules that contain at least one of these blocking buckets. As a result, any comparison of pairs of records will be performed on subsets of the cartesian product that satisfy the blocking conditions. The candidate rules are then compiled into Spark join operations that can run at scale.

2.3 Explainable Active Learning

`SystemER` provides different ways to start active learning: (1) learning from scratch either with or without labeled data, (2) starting with a manual rule (see Figure 4c) if the user has a candidate rule in mind, and (3) continue from a previously finished iteration, where the user stopped last time.

The active learning phase involves multiple iterations, where each iteration includes the following steps (see Figure 3):

1. `SystemER` produces a candidate rule `R` based on the available labeled data.
2. To verify `R`, `SystemER` selects a few examples that satisfy `R` but have low-confidence scores. We call these examples *likely false positives* (LFP) and have the user label them.
3. The user can either accept `R` if most (e.g., 90%) of the LFPs turn out to be matches, or reject it otherwise.
4. The system will then search for examples that do not satisfy `R` but are likely to be matches with high-confidence scores (the exploration is done by relaxing `R`). We call these examples *likely false negatives* (LFN), and the user also needs to label them.
5. Update labeled data with the newly labeled examples.

The two types of examples selected in steps 2 and 3 guide the system to refine the ER rules in different ways: (1) Low-confidence LFPs can be seen as adversarial examples to falsify the current rule, and thus can potentially improve the precision of the subsequent learned rule. (2) High-confidence LFNs are important for improving the overall recall of the ER model. Recall that `SystemER` considers only features that measure goodness of match. Therefore, the confidence score of an entity-pair `p` can be measured by the number of features that evaluate to true on `p`. More technical details of the adopted active learning algorithm can be found at [8].

Since both the intermediate rules and final model produced by `SystemER` are explainable, it can help the user understand the labeling process by offering insights into the active learning process. We carefully designed the labeling interface (Figure 4d) so that it is simple enough to hide the technical complexity but still achieve high explainability by providing evidence why those examples are selected for labeling. Moreover, matching functions of a rule are clickable. Once a function (e.g., `dblp.year=acm.year` in Figure 4d) is clicked, the corresponding attribute pairs (i.e., `year`) will be highlighted to help the user understand why this example satisfies the function and the rule.

Start without labeled data. When no labeled data is provided, `SystemER` will use one of the blocking buckets to form the first candidate rule `C` to start the active learning. Then, in the first iteration, examples that satisfy `C` with low-confidence (resp. high-confidence) score will be used as likely false positives (resp. likely false negatives).

Quality guarantee. The learning philosophy of `SystemER` is that a candidate rule will be accepted only if it achieves high precision in likely false positives, which is a conservative way to conclude that the rule is high-precision [8]. Multiple active learning iterations are desired so that a variety of semantically different high-precision rules can be learned, which collectively achieve high recall. `SystemER` maintains a repository that keeps all the accepted rules so that when searching for new examples to be labeled, examples that are already covered by the accepted rules will be excluded, which makes the active learning more effective.

Empirical study. A thorough empirical study reported in [8] shows that the active learning methodology adopted in `SystemER` can significantly outperform both statistical models (e.g., support vector machines) and models that combine statistics with first-order logic rules (e.g., learning-based Markov Logic Networks or MLN [9]) across various matching tasks. Specifically, in a big data scenario (matching 470K employee records with 50M social profiles), `SystemER` produces 7 rules that can find 10 times more correct matches than MLN under the same precision constraint, while using the same number of user labels as MLN.

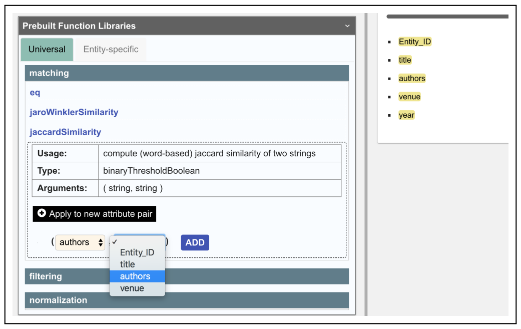
2.4 Scalability and Optimizations

`SystemER` expresses ER rules in the language of HIL [3], which can be compiled into different high-performance runtimes. However, unlike the prototype system in [8] that runs on MapReduce only, `SystemER` now can distribute the computation via a Spark cluster, which is more efficient. For small datasets (e.g., DBLP-ACM), single-node mode is sufficient, but for large datasets (e.g., twitter users vs. Walmart customers) cluster mode can significantly speed up computation. Specifically, one of our realistic matching tasks aims to identify matches across two company datasets (one has 20K records and the other has 12M records). With single-node mode, one iteration may take more than two hours. However, with a cluster of 10 nodes, the running time reduced to 20-30 minutes including labeling time.

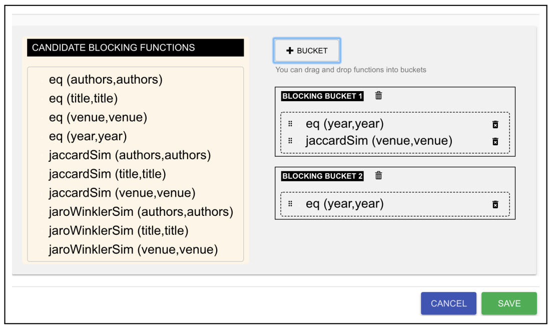
3. DEMO SCENARIOS

`SystemER` provides several predefined scenarios including DBLP-ACM (bibliography datasets with flat records), and Pubmed[6] (matching of authors across biomedical publication

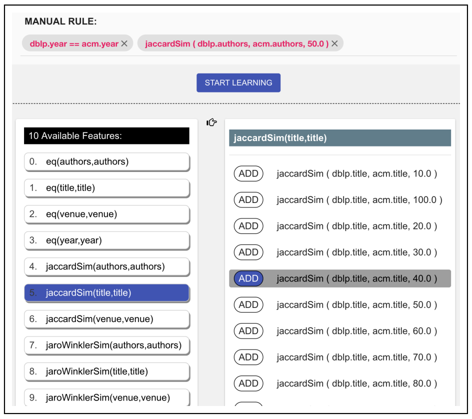
[6]`https://www.ncbi.nlm.nih.gov/pubmed/`



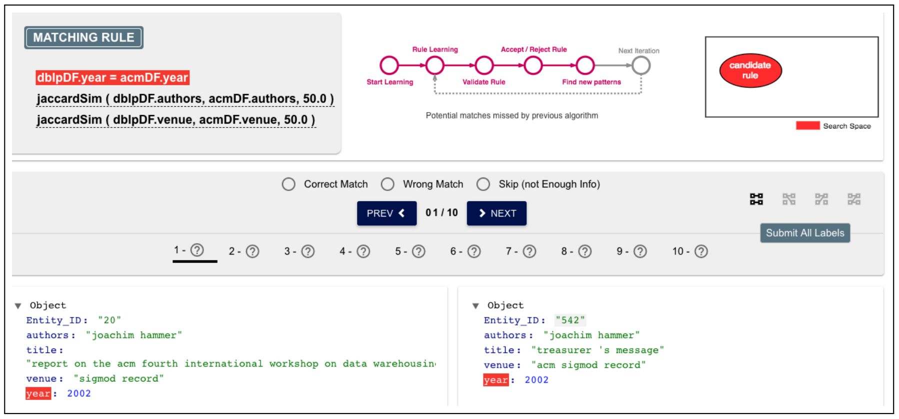
(a) Feature engineering



(b) UI for setup blocking functions


MANUAL RULE:
dblp.year == acm.year
jaccardSim (dblp.authors, acm.authors, 50.0)
START LEARNING
10 Available Features:
0. eq(authors,authors)
1. eq(title,title)
2. eq(venue,venue)
3. eq(year,year)
4. jaccardSim(authors,authors)
5. jaccardSim(title,title)
6. jaccardSim(venue,venue)
7. jaroWinklerSim(authors,authors)
8. jaroWinklerSim(title,title)
9. jaroWinklerSim(venue,venue)
jaccardSim(title,title)
ADD jaccardSim (dblp.title, acm.title, 10.0)
ADD jaccardSim (dblp.title, acm.title, 100.0)
ADD jaccardSim (dblp.title, acm.title, 20.0)
ADD jaccardSim (dblp.title, acm.title, 30.0)
ADD jaccardSim (dblp.title, acm.title, 40.0)
ADD jaccardSim (dblp.title, acm.title, 50.0)
ADD jaccardSim (dblp.title, acm.title, 60.0)
ADD jaccardSim (dblp.title, acm.title, 70.0)
ADD jaccardSim (dblp.title, acm.title, 80.0)

(c) Start with a manual rule



```
▼ Object
  Entity_ID: "20"
  authors: "joachim hammer"
  title:
  "report on the acm fourth international workshop on data warehousin
  venue: "sigmod record"
  year: 2002
```

```
▼ Object
  Entity_ID: "542"
  authors: "joachim hammer"
  title: "treasurer 's message"
  venue: "acm sigmod record"
  year: 2002
```

(d) Labeling interface

Figure 4: Screenshots of `SystemER` in action

records with nested structures). The demo attendees can either provide their own data or use one of the provided scenarios. Taking DBLP-ACM as an example, the user will experience the following steps:

1. the user provides `dblp.csv` and `acm.csv` to `SystemER`, and the `SystemER` will automatically infer the schemas to create a new scenario.
2. The user can apply various matching functions (e.g., `equality`, `Jaccard` and `JaroWinkler` similarity) over different attribute pairs of the two datasets.
3. The user can specify blocking conditions, for example, `dblp.year=acm.year`.
4. The user can learn from scratch (without labeled data).
5. The system will learn a candidate rule (e.g., the one shown in Figure 4d) together with 10 likely false positives and 8 likely false negatives. The user then needs to label them and continue to the next iteration.
6. The system will learn a new rule (such as the rule shown in Figure 1), which refines the previous one by either adding an addiontal predicate or changing the threshold values for some similarity functions. If the new rule is precise, the user can choose to accept it. Otherwise, the user can move to the next iteration.

The user can continue the learning process for multiple iterations to see how `SystemER` refines the rules to produce subsequent good rules. Additionally, the user can run `SystemER` in different execution modes (single-node vs. cluster).

4. REFERENCES

[1] N. Bhutani, K. Qian, Y. Li, H. V. Jagadish, M. A. Hernández, and M. Vasa. Exploiting structure in representation of named entities using active learning. In *COLING*, 2018.

[2] Z. Chen, Q. Chen, and Z. Li. A human-and-machine cooperative framework for entity resolution with quality guarantees. In *ICDE 2017*, pages 1405–1406.

[3] M. A. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. Hil: a high-level scripting language for entity integration. In *EDBT*, 2013.

[4] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(13):1197–1208, 2016.

[5] P. Konda, S. Das, S. GC, et al. Technical perspective:: Toward building entity matching management systems. *ACM SIGMOD Record*, 47(1):33–40, 2018.

[6] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.

[7] K. Qian, N. Bhutani, Y. Li, H. Jagadish, and M. Hernandez. Lustre: An interactive system for entity structured representation and variant generation. In *ICDE 2018*, pages 1613–1616.

[8] K. Qian, L. Popa, and P. Sen. Active learning for large-scale entity resolution. In *CIKM*, pages 1379–1388, New York, NY, USA, 2017. ACM.

[9] P. Singla and P. Domingos. Entity resolution with markov logic. ICDM'06, pages 572–582.