CloudMatcher: A Hands-Off Cloud/Crowd Service for Entity Matching

Yash Govind[1], Erik Paulson[1,2], Palaniappan Nagarajan[1], Paul Suganthan G.C.[1], AnHai Doan[1], Youngchoon Park[2], Glenn M. Fung[3], Devin Conathan[3], Marshall Carter[3], Mingju Sun[3]

[1]University of Wisconsin-Madison, [2]Johnson Controls, [3]American Family Insurance

{ygovind, paln, anhai, paulgc}@cs.wisc.edu,{erik.paulson, youngchoon.park}@jci.com, {gfung, dconatha, mcarter1, msun}@amfam.com

ABSTRACT

As data science applications proliferate, more and more lay users must perform data integration (DI) tasks, which used to be done by sophisticated CS developers. Thus, it is increasingly critical that we develop hands-off DI services, which lay users can use to perform such tasks without asking for help from developers. We propose to demonstrate such a service. Specifically, we will demonstrate CloudMatcher, a hands-off cloud/crowd service for entity matching (EM). To use CloudMatcher to match two tables, a lay user only needs to upload them to the CloudMatcher's Web page then iteratively label a set of tuple pairs as match/no-match. Alternatively, the user can enlist a crowd of workers to label the pairs. In either case, the lay user can easily perform EM end-to-end without having to involve any developers. CloudMatcher has been used in several domain science projects at UW-Madison and at several organizations, and is scheduled to be deployed in a large company in Summer 2018. In the demonstration we will show how easy it is for lay users to perform EM (either via interactive labeling or crowdsourcing), how users can easily create and experiment with a range of EM workflows, and how CloudMatcher can scale to many concurrent users and large datasets.

PVLDB Reference Format:
Y. Govind, E. Paulson, P Nagarajan, Paul S. G.C., AnHai Doan, Y. Park, G. M. Fung, D. Conathan, M. Carter, M. Sun. CloudMatcher: A Hands-Off Cloud/Crowd Service for Entity Matching. *PVLDB*, 11 (12): 2042-2045, 2018.
DOI: https://doi.org/10.14778/3229863.3236255

1. INTRODUCTION

Entity Matching (EM) finds data instances that refer to the same real-world entity, such as the two tuples (David Smith, UW-Madison) and (D. Smith, UWM). This problem has been a long-standing challenge in data management, and numerous solutions have been developed [5, 2, 4, 9].

While much progress has been made, current solutions are still limited in that they often *require a developer* to be involved in the matching process. For example, several recent solutions require a developer to write heuristic rules, called *blocking rules*, to reduce the number of candidate tuple pairs to be matched, then train and apply a matcher to the remaining pairs to predict matches. The developer must know how to code (e.g., to write rules in Python) and match entities (e.g., to select learning models and features).

As such, these solutions do not work well for *lay users*, such as domain scientists (e.g., economists, zoologists), business teams at companies, journalists, and data enthusiasts. These users often have little or no EM knowledge (e.g., they do not know blocking and string similarity measures), and thus cannot act as developers. Yet, as the number of data science applications proliferates in more and more domains, more and more such users will have to perform EM. Consequently, it is increasingly critical that we develop EM solutions that are very easy for lay users to use.

Hands-Off Entity Matching: To address this problem, in recent work [6, 3] we have introduced the idea of *hands-off entity matching*, where a lay user can easily perform EM end-to-end, without having to involve a developer. Specifically, to match two tables A and B, the user only needs to label a set of tuple pairs $(a \in A, b \in B)$ as match/no-match. The system uses these pairs to infer blocking rules, perform blocking, learn a matcher, then apply the matcher to produce the matches. Alternatively, the user can just ask a crowd of workers (e.g., a team of collaborators or workers on Mechanical Turk) to label the tuple pairs. The paper [6], which describes the Corleone system, shows that this approach is highly promising. A subsequent paper [3], which describes the Falcon system, shows how to scale Corleone to efficiently match large tables (e.g., of millions of tuples).

The CloudMatcher Service: The works Corleone and Falcon only develop an algorithmic solution. They do not build a complete end-to-end industrial-strength hands-off EM system. In the past two years, building on the above works, we have sought to build a system like that, called CloudMatcher. To use CloudMatcher, a lay user goes to *cloudmatcher.io* (not yet open to the public, but will be

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236255

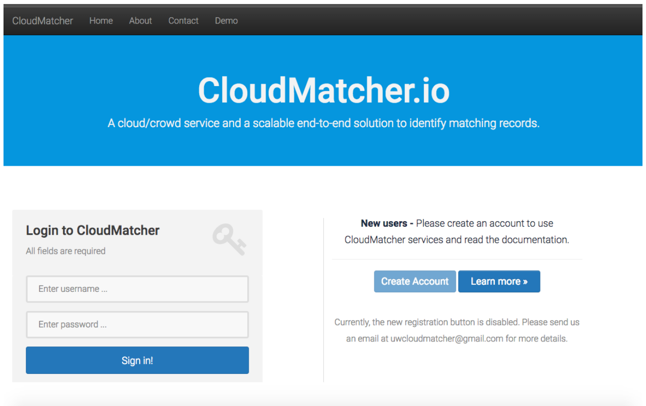

Figure 1: The front page of the CloudMatcher service.

in the near future, see Figure 1), creates an account, and uploads two tables to be matched. The user then iteratively labels a set of tuple pairs, or enlists a set of workers to label, in a crowdsourcing fashion. As such, CloudMatcher is a *hands-off cloud/crowd service for entity matching*. Our goal is threefold: (1) providing CloudMatcher as an EM service for UW-Madison domain scientists, corporate partners, and the general public, (2) open sourcing CloudMatcher so that anyone can deploy this EM service in-house, and (3) using CloudMatcher to evaluate and drive our research on hands-off EM.

While still under heavy development, CloudMatcher has proven to be highly promising. Version 1.0 of CloudMatcher has been used in several domain science projects at UW-Madison, and at several organizations (e.g., Johnson Controls, Marshfield Clinic, and a non-profit organization). In addition, the Fortune-500 American Family Insurance Inc. has joined the project as a funder and developer, and planned to deploy a version of CloudMatcher in Summer 2018, to help their business teams perform EM. The following example illustrates the promise of CloudMatcher.

Example 1. *In Spring 2017, a team of economists at UW-Madison led by Dr. Brent Hueth had to match two tables of US organizations. They hired a CS graduate student as a developer. It took this student more than a week to learn about EM (e.g., how to perform blocking, how to match using supervised learning, etc.) and match the above two tables, and yet he had not been able to produce the matches. At that time, since Version 1.0 of* CloudMatcher *just became operational, we asked that team to use* CloudMatcher. *The team spent under 50 minutes to label 680 tuple pairs, and obtain the matches in 61 minutes (this includes 11 minutes of machine time), achieving a precision of 92% and recall of 96%. Thus,* CloudMatcher *helped perform EM in an hour, instead of days or possibly weeks with the graduate student.*

As described, CloudMatcher is highly promising. Developing it, however, raises many novel challenges. First, we need to scale CloudMatcher to handle many concurrent EM workflows, e.g., submitted by many users. (Falcon only scales up the execution of a single EM workflow.) Second, we need to handle fault tolerance and crash recovery. Third, we need to design the system in a modular and extensible fashion, to facilitate new features and debugging. Finally, we want the system to provide not a rigid EM workflow (as Falcon does, see Section 2), but a set of basic services, so that users can

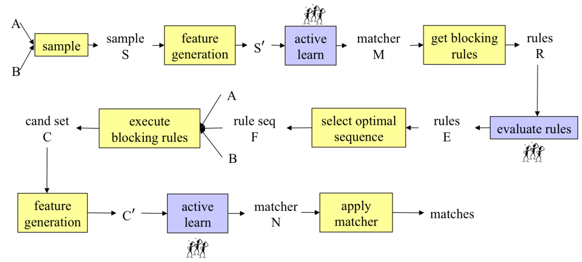

Figure 2: The EM workflow of Falcon.

use these services to easily customize and experiment with a range of EM workflows.

We have addressed some of these challenges in a recent workshop paper [7] and have completed the development of Version 2.0 of CloudMatcher. (This version however still points to many interesting R&D challenges to be addressed in future work.) Here we propose to demonstrate this version. Our goals are as follows. First, we will demonstrate the idea of *hands-off data integration (DI) services*. As data science explodes and more lay users must do DI, we believe hands-off DI services, where lay users can perform a DI task end-to-end without involving a developer, will become increasingly critical. In fact, companies such as Amazon have been talking about similar ideas called "hands-off-the-wheel data services" (that can easily be used by their business teams without involving their developers). As far as we can tell, this proposal is the first to demonstrate a hands-off cloud/crowd DI service for entity matching.

Second, we will demonstrate the promise of hands-off EM, by showing how easy it is for lay users to perform EM, either by interactive labeling, or by using crowd to label.

Third, we will show how users can easily customize and experiment with a range of EM workflows, by combining basic services provided by CloudMatcher on an easy-to-use cloud-based UI. Finally, we will show how CloudMatcher can scale to many concurrent users and big data sets, using a cluster of machines in the backend.

Related Work: Numerous EM solutions have been developed (see [5, 2, 4] for surveys and books). These solutions are limited in that they often require a developer in the loop. Two recent works [6, 3] propose hands-off EM, which a lay user can perform without involving a developer. CloudMatcher leverages these works to build an end-to-end industrial-strength cloud/crowd EM service. This raises many novel challenges, as described earlier, some of which have been discussed in a recent workshop paper [7]. CloudMatcher has not been demonstrated before at a database conference, and is the first academic work to build such a service, as far as we know. In industry, we know of only one other similar work, Dedupe [1], which is a cloud-based EM service to match tuples within a single table. Dedupe, however, uses only simple types of blockers and requires the user to label tuple pairs using active learning. In contrast, CloudMatcher can employ crowdsourcing to label the pairs (CloudMatcher also supports the user mode). As far as we can tell, CloudMatcher is the first cloud-based EM service that provides support for crowdsourcing. It is also not clear from the public documentation whether Dedupe can scale to many concurrent users and large tables. See [7] for a more detailed discussion of related work, including crowdsourcing EM, building EM systems, and scaling EM.

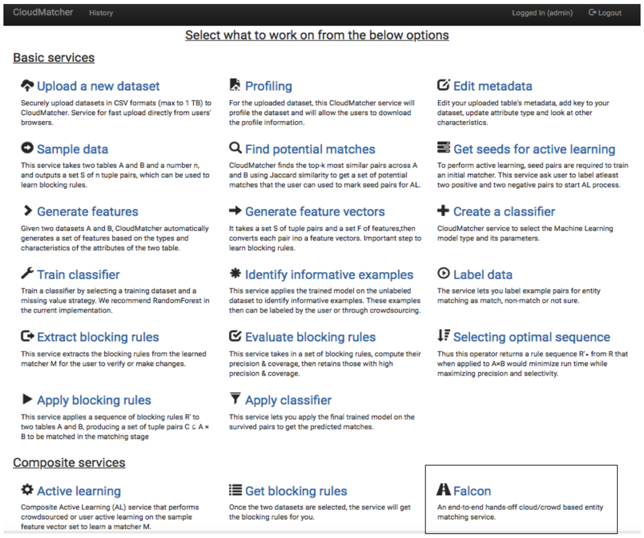

Figure 3: The services of CloudMatcher.

2. THE CLOUDMATCHER SERVICE

We now briefly describe CloudMatcher, focusing only on aspects important for this demonstration. In the default mode, when a user uploads the two tables A and B to be matched, CloudMatcher executes the Falcon EM workflow shown in Figure 2. It first takes a sample of S tuple pairs from A and B, and converts each pair into a feature vector, producing a set S' of feature vectors. Next, it performs active learning on S' to learn a matcher M, which is a random forest. CloudMatcher then extracts candidate blocking rules from M, evaluates the rules, then selects an optimal rule sequence F. Next, it performs blocking, i.e., executing F on tables A and B, to obtain a set of candidate tuple pairs C. Finally, it performs active learning on C to obtain a new matcher N, applies N to C to predict match/no-match, then outputs the matches ([3, 7] describe this workflow in detail).

Note that in the above workflow the lay user interacts with CloudMatcher in only three places (labeled in light blue): active learning to get a matcher M, evaluating blocking rules, and active learning to get a matcher N. In all three places, the user only needs to label a set of tuple pairs as match/no-match. No other knowledge and action are necessary. The paper [7] describes how we implement the above workflow in CloudMatcher, including how we address challenges such as scaling up to multiple concurrent EM workflows, and handling fault tolerance and crash recovery.

Version 1.0 of CloudMatcher implemented only the above Falcon EM workflow. As we interacted with real users, however, we observed that many users want to flexibly customize and experiment with different EM workflows. As a result, in Version 2.0, we solved this problem by (a) extracting a set of basic services from the Falcon EM workflow and making them available on CloudMatcher, and (b) allowing users to flexibly combine them to form different EM workflows, including the original Falcon one. Figure 3 shows examples of services that we currently provide. *Basic services* include uploading a dataset, profiling a dataset, edit the metadata of a dataset, sampling, generating features, training a classifier, etc. We have combined these basic services to provide *composite services*, such as active learning, obtaining blocking rules, and Falcon (see the bottom of the figure). For

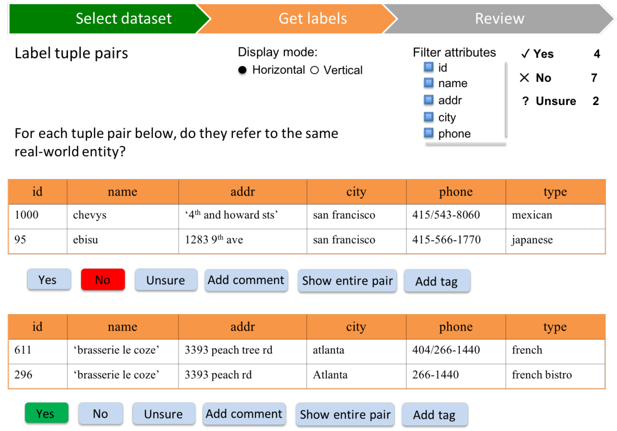

Figure 4: Labeling tuple pairs in the user mode.

example, the user can invoke the "Get blocking rules" service to ask CloudMatcher to suggest a set of blocking rules that he/she can use. As another example, the user can invoke the "Falcon" service to execute the end-to-end Falcon EM workflow. We discuss using these services more in the next section.

3. DEMONSTRATION OVERVIEW

We now describe the proposed demonstration. We will show that (a) it is easy for a lay user to perform EM on CloudMatcher, end-to-end, via interactive labeling, or by using a crowd of workers; (b) the user can easily customize and experiment with a wide range of EM workflows, by combining CloudMatcher services; and (c) CloudMatcher can scale to many concurrent users and big data sets, using a cluster of machines in the backend.

We will focus on the scenario of matching two tables. In practice, performing EM often takes tens of minutes or hours. So a large part of our demonstration (e.g., labeling multiple tuple pairs) will be "canned" scenarios. But we will provide opportunities for the audience to interact "live" with CloudMatcher, and to take the demo "off the rails".

3.1 How Lay Users Can Easily Perform EM with CloudMatcher

We will show a scenario where a lay user easily uploads two tables to be matched to CloudMatcher, profiles it, edits its metadata if necessary, then interactively labels tuple pairs to perform EM. Figure 4 shows the current labeling interface of CloudMatcher. In particular, it shows two tuple pairs that the user can label as yes/no/unsure, and so on.

We will also show a scenario where the lay user enlists a crowd of workers to label tuple pairs (we will simulate this crowd of workers, or ask the audience to participate as a crowd of workers). This scenario will demonstrate that with crowdsourcing, performing EM on CloudMatcher is as easy as uploading the tables, doing some pre-processing, then providing a credit card to pay for the crowd.

We will show a scenario where lay users do not have access to a hands-off service such as CloudMatcher, and must use instead an EM tool such as Magellan to perform EM. Magellan [8] is a state-of-the-art end-to-end EM system that we have developed (in Python). It is geared toward power users (e.g., those who know how to code and perform EM). We will

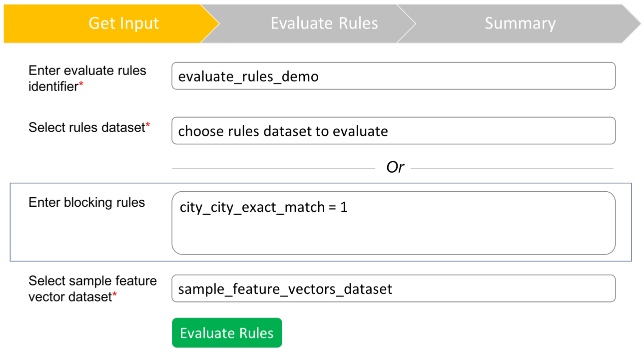

Figure 5: How a user enters a blocking rule to be evaluated.

show how Magellan works by showing Jupyter notebooks of its sessions (taken from real-world matching scenarios), show how difficult it is for lay users to perform such a session, and then contrast that with the ease of using CloudMatcher.

3.2 How Lay Users Can Create and Experiment with Many EM Workflows

The above scenarios show how CloudMatcher executes the Falcon EM workflow (see Section 2). In practice, as mentioned earlier, while many users are satisfied with executing just this workflow, many other users want to create and experiment with different EM workflows. In this part of the demonstration, we will show two scenarios to demonstrate that lay users can indeed do so, by combining the basic services of CloudMatcher in different ways. We will also ask the demonstration attendees to try to create different kinds of EM workflows, using the services of CloudMatcher.

Scenario 1: We will demonstrate a scenario in which a user wants to match two tables of restaurant descriptions. Earlier, to do so, CloudMatcher would have started by asking the user to label tuple pairs so that it can learn a set of blocking rules. Here, however, suppose that the user already knows a good blocking rule, namely, R_1 = "if two restaurants disagree on the value of the city attribute, then they will not match and hence should be blocked". Thus, the user wants to skip the step of learning blocking rules, and use the blocking rule R_1.

However, just because the user thinks R_1 is a good rule does not mean it is indeed a good rule. It can be a bad rule for multiple reasons, such as many values of city are missing or misspelled. As a result, the user first invokes the basic service "Evaluate Blocking Rules", and enters rule R_1 (see the screen shot in Figure 5). CloudMatcher then asks the user to interactively label a set of tuple pairs, and uses these labeled pairs to evaluate the quality of R_1. If R_1 turns out to be a good blocking rule, then the user can invoke the basic service "Apply blocking rules" to perform blocking using R_1, then invoke other basic services to learn a matcher and apply the matcher. Thus, this EM workflow differs from the Falcon workflow in that here the user directly supplies a blocking rule (rather than learning it, as in Falcon).

Scenario 2: To continue with the above scenario, let C be the set of tuple pairs obtained after applying the blocking rule R_1 to the two input tables. Earlier, executing the Falcon EM workflow, CloudMatcher would have interacted with the user in an active learning fashion on C to learn a matcher M, then apply M to C. However, suppose the user does not want to perform active learning. Rather, the user wants to take a random sample S from C, labels S using crowdsourcing, splits S into two sets I and J, uses I to train a matcher M, applies M to J to compute precision and recall on J, then uses these numbers to estimate precision and recall on C. (The paper [6] shows how to perform this estimation, and the Magellan EM system [8] supports such EM workflows.) We will show how the user can invoke the basic services of CloudMatcher to execute the above workflow.

3.3 How CloudMatcher Scales to Many Users and Big Datasets

For CloudMatcher to be truly useful in practice, it must scale to many users and big datasets. In the last part of the demonstration, we will demonstrate CloudMatcher in these aspects. We will run a simulation that generates many users who concurrently run EM workflows on CloudMatcher. We will use CloudMatcher's monitoring dashboards to show how the system copes as the load increases, and how the execution of the EM workflows progresses. We will also use CloudMatcher's dashboard to show how the system executes matching two large tables (in the range of 1.2M-2.5M tuples) on a cluster of machines.

By the time of the demonstration, if the system has been successfully deployed and used at American Family Insurance, we will also obtain permission to show its dashboards, thereby showing how many users are using the system and how it executes their tasks.

4. CONCLUSIONS

We argue that as more and more lay users must perform DI tasks, it is important that we develop hands-off DI services, which they can use without asking for help from the CS developers. Toward this goal, in this demonstration we will present CloudMatcher, a hands-off cloud/crowd service for entity matching. As far as we can tell, no such hands-off service has been demonstrated for entity matching. We focus on showing (a) it is easy for a lay user to perform EM on CloudMatcher, via interactive labeling, or by using a crowd of workers; (b) the user can easily create a wide range of EM workflows, by combining CloudMatcher services; and (c) CloudMatcher can scale to many concurrent users and big data sets, using a cluster of machines in the backend.

5. REFERENCES

[1] Dedupe https://dedupe.io/.
[2] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer Publishing Company, Incorporated, 2012.
[3] S. Das et al. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, 2017.
[4] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration.* Morgan Kaufmann, 1st edition, 2012.
[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007.
[6] C. Gokhale et al. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.
[7] Y. Govind et al. Cloudmatcher: A cloud/crowd service for entity matching. In *BIGDAS*, 2017.
[8] P. Konda et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.
[9] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.