# Executing Entity Matching End to End: A Case Study

Pradap Konda[1], Sanjay Subramanian Seshadri[2], Elan Segarra[3], Brent Hueth[3], AnHai Doan[3]
[1]Facebook, [2]NetApp, [3]University of Wisconsin-Madison, U.S.A.

## ABSTRACT

Entity matching (EM) identifies data instances that refer to the same real-world entity. Numerous EM works have covered a wide spectrum, from developing new EM algorithms to scaling them to building EM systems. But there has been very little if any published work on *how EM is carried out in practice, end to end*. In this paper we describe in detail a case study of applying EM to a particular domain end to end (i.e., going from the raw data all the way to the matches).

Specifically, we describe a real-world application for EM in the science policy research community. We describe how our team (the EM team) interact with the science policy team to carry out the EM process, using PyMatcher, a state-of-the-art EM system developed in the Magellan project at UW-Madison. We highlight the communication between the two teams and the zig-zag nature of the EM process. We identify a set of challenges that we believe arise in many real-world EM projects but that current EM systems have either ignored or are not even aware of. Finally, we provide all data underlying this case study, including labeled tuple pairs and documentation supplied by the science policy team, to serve as a good challenge problem for EM researchers.

## 1 INTRODUCTION

Entity matching (EM) identifies data instances that refer to the same real-world entity, such as (David Smith, UW-Madison) and (D. M. Smith, UWM). EM has been a long-standing challenge in data management [6, 11], and will become even more important in data science. This is because many data science projects need to integrate data from disparate sources before analysis can be carried out, and such integration often requires EM.

Consequently, EM has received enormous attention (see Section 2). Surprisingly, as far as we can tell, there has been very little if any published work on *how EM is carried out in practice, end to end*. The closest that we can find are works that perform EM in a particular domain, e.g., e-commerce, mobile data, patient records, drugs, etc. But these works focus on developing specialized EM algorithms that exploit the characteristics of the target domain, e.g., exploiting product taxonomies (to match e-commerce products) or the spatio-temporal nature of mobile data (to match phone calls).

In this paper we describe in detail a case study of applying EM to a particular domain, end to end (i.e., going from the raw data all the way to the matches). This case study is quite rich, *and it clearly demonstrates many novel challenges for current EM solutions and systems.* We will soon release all the data underlying this case study (to be available at [20]), so that our community can use it as a challenge problem for EM.

Specifically, in summer 2015 we started the Magellan project at the University of Wisconsin-Madison, to build EM systems. We believe that building practical EM systems is critical for advancing the EM field, the way systems such as System R, Ingres, Hadoop, and Spark are critical for advancing the fields of relational data management and Big Data. Subsequently, we built an EM system called PyMatcher [17], which is quite different from EM systems built so far (see Section 2). We then looked for real-world applications to "test drive" PyMatcher.

To do so, we talked with the science policy research community, which has been building a large "data lake" called UMETRICS. Currently UMETRICS collects data from 24 participating U.S. universities, on research grants and the researchers involved in these grants. They use the data to study questions such as "What are the results of investments in research?" and "How do universities affect the regional economy?", and indeed many studies have been published using UMETRICS data (e.g., [30]).

Building UMETRICS requires matching grants across different datasets. The UMETRICS team (at the University of Michigan) has developed and deployed a rule-based EM workflow, but its accuracy is not satisfactory. So we collaborated with a science policy team at UW-Madison to build a more accurate EM workflow, using PyMatcher.

This work was eye-opening to us. Before starting this work, we already felt that existing EM systems were not powerful enough because they failed to address many challenges that arise in real-world EM [17]. That was why we started the Magellan project. Working with UMETRICS suggested to us that we were probably on the right track, but that real-world EM does raise many more challenges that we have also failed to recognize.

These challenges apply equally well to Magellan and other existing EM systems. We discuss the challenges in detail in Section 13, covering the need to develop how-to guides (that provide detailed guidance to the users on how to carry out EM step by step, end to end), new pain points in EM (e.g., labeling, match definition), the need for different EM solutions for different parts of the data, support for easy collaboration among multiple team members (who are often in different locations), handling changes that inevitably arise during the EM process, managing machine learning "in the wild", the need to use both learning and hand-crafted rules, and the need to design a new type of EM architecture. In summary, we make the following contributions:

- We describe a real-world application for EM in the science policy community. We describe what the users want in terms of EM, how their goals change over time, and what end results they are aiming for.
- We describe in detail how our team (the EM team) interacts with the science policy team (the UMETRICS domain expert team) to carry out the EM process, using PyMatcher, a state-of-the-art EM system developed in our group. We highlight the communication between the two teams and the zig-zag nature of the EM process.
- We identify a set of challenges that we believe arise in many real-world EM projects but that current EM systems have either ignored, or are not even aware of, or have not addressed well. While in this paper we have focused on just one case study, in the past few years we have worked with many more real-world EM cases, and the challenges described here also commonly arise in those cases.

- Finally, we provide all data underlying this case study, including all the labeled tuple pairs and documentation supplied by the domain expert [20]. This dataset can serve as a good challenge problem for EM researchers.

Overall, we hope that the case study here can contribute to helping EM researchers understand better the challenges of the EM process and develop more effective EM solutions. The UMETRICS data and Jupyter notebooks will be available at [20], and a technical report with far more details will be available at [19].

## 2 BACKGROUND & RELATED WORK

We now discuss the EM problem, related work, then PyMatcher, which was used to perform EM for the case study.

**Entity Matching:** This problem, a.k.a. entity resolution, record linkage, etc., has received enormous attention (see [6, 11] for books and surveys). A common EM scenario finds all tuple pairs $(a, b)$ that match, i.e., refer to the same real-world entity, between two tables $A$ and $B$ (see Figure 1). Other EM scenarios include matching tuples within a single table, matching into a knowledge base, matching XML data, etc. [6].

Most EM works develop matching algorithms, exploiting rules, learning, clustering, crowdsourcing, among others [6, 11]. They try to improve the matching accuracy and reduce costs (e.g., run time). Trying to match all pairs in $A \times B$ often takes very long. So users often employ heuristics to remove obviously non-matched pairs (e.g., products with different colors), in a step called *blocking*, before matching the remaining pairs. Several works have addressed scaling up blocking (e.g., [3, 8, 16, 29]), learning blockers [4, 9], and using crowdsourcing for blocking [13] (see [7] for a survey).

Many works have considered EM for a particular domain, such as e-commerce (e.g., [10, 15, 21, 22]), patient records [24], financial entities [12], medical sciences [5], drugs [26], and more. But they do not focus on the entire end-to-end process, the interaction between the EM team and the domain expert team, and on identifying the challenges for current EM systems, as we do in this paper.

In contrast to the extensive effort on EM algorithms, there has been relatively little work on building EM systems. As of 2016 we counted 18 major non-commercial systems (e.g., D-Dupe, DuDe, Febrl, Dedoop, Nadeef), and 15 major commercial ones (e.g., Tamr, Data Ladder, IBM InfoSphere) [6]. Our examination of these systems (see [18]) reveals the following four major problems: these systems do not cover the entire EM pipeline, they are stand-alone monolithic systems and hence they make it very difficult to exploit a wide range of techniques, it is very difficult to write code to "patch" these systems even though this is often necessary in practice, and these systems provide very little guidance to users on how to execute the EM process. These shortcomings motivated us to develop the Magellan project, which we briefly describe next.

**The Magellan Project (PyMatcher and CloudMatcher):** In the Magellan project we developed an on-premise Python-based EM system called PyMatcher and a cloud-based self-service system called CloudMatcher. In this paper we only used the PyMatcher system, which we briefly describe below.

Compared to current EM systems, PyMatcher is novel in four aspects. (1) It provides how-to guides that tell users what to do in each EM scenario, step by step. (2) It provides tools to help users do these steps; the tools seek to cover the entire EM pipeline, not just matching and blocking as current EM systems do. (3) Tools are built into the ecosystem of data science tools in Python, allowing PyMatcher to borrow powerful capabilities in data cleaning, visualization, learning, etc. (4) PyMatcher provides a powerful scripting environment to facilitate interactive experimentation and quick "patching" of the system. See [14, 17] for more, and see *sites.google.com/site/anhaidgroup/projects/magellan* for code.

**Table A**

| | Name | City | State |
|---|---|---|---|
| $a_1$ | Dave Smith | Madison | WI |
| $a_2$ | Joe Wilson | San Jose | CA |
| $a_3$ | Dan Smith | Middleton | WI |

**Table B**

| | Name | City | State |
|---|---|---|---|
| $b_1$ | David D. Smith | Madison | WI |
| $b_2$ | Daniel W. Smith | Middleton | WI |

| Matches |
|---|
| $(a_1, b_1)$ |
| $(a_3, b_2)$ |

**Figure 1: An example of matching two tables.**

## 3 PROBLEM DEFINITION

We now introduce the science policy research community, their effort to conduct data-driven research by creating a large data lake called UMETRICS, their need to perform EM to build UMETRICS, and the EM problem considered in this paper.

**The Science Policy Research Community:** A large country such as the U.S. spends hundreds of billions of dollars on science R&D (i.e., research & development) per year. It is important to be able to track the impact of this spending and develop effective science R&D policies. The science policy research community studies such issues.

**UMETRICS:** To do so effectively, in the past decade, this community has been building a data science infrastructure centering around a large data lake called UMETRICS, which stands for "Universities: Measuring the Impacts of Research and Innovation, Competitiveness and Science" [2].

Currently UMETRICS collects data from 24 participating universities, specifically data on (a) all people paid and all purchases from vendors and subcontracts for all federally funded grants, and (b) the specific job titles and the source of funding for research projects at the universities. As such, the data can be used to study questions such as "What are the results of investments in research?" and "How do universities affect the regional economy?", and indeed many studies have been published using UMETRICS data (e.g., [30]).

**The Need for Entity Matching:** While highly promising, building UMETRICS require a lot of work to integrate the data submitted by the 24 participating universities. For example, if UW-Madison submits a table that lists research grants from the National Science Foundation (NSF), then those grants must be matched into existing grants already in UMETRICS.

This matching is highly non-trivial. For example, the same research project can have different research titles recorded in UMETRICS and at universities. As another example, a grant given by a funding agency to a research project may be distributed to many smaller projects in the same university, and this information will be recorded in multiple entries in UMETRICS, so matching these entries is not trivial.

**The Need to Improve the Current EM Solution:** Currently the UMETRICS team (at the University of Michigan) performs such matching using hand-crafted rules. But the accuracy remains unsatisfactory. As a result, a team led by Professor Brent Hueth (in the Department of Agricultural and Applied Economics at UW-Madison and a co-author of this paper) set out to examine how to improve the accuracy of this EM solution.

**Matching the USDA Dataset with the UMETRICS Dataset:** As a concrete project, they first selected a dataset that contains grants awarded to UW-Madison from USDA (the U.S. Department of Agriculture) in a certain time period. We will refer to this dataset the USDA dataset.

Next, they wanted to match grants in this USDA dataset into UMETRICS. To do so, they performed a simple selection in UMETRICS to select all grants awarded to UW-Madison by USDA in the same time period (UMETRICS allows such selections). For simplicity, let us call this new dataset the UMETRICS dataset. *Next, the team wanted to match grants between the USDA dataset and the* UMETRICS *dataset. Their goal is to find a new EM workflow that is more accurate than the current rule-based EM workflow (deployed in* UMETRICS*).*

This team, which we will call the UMETRICS team, consisted of Professor Brent Hueth, a Ph.D. student in economics, and an hourly student in CS. Initially, we did not get closely involved in the EM process. Instead, we let the UMETRICS team try to match the two datasets themselves. We only provided advice for specific problems such as debugging the matches and evaluating the match results. However, after multiple meetings, we observed that the UMETRICS team simply did not know how to perform EM in a systematic fashion. They had no idea what to do first, what to do second, etc., even though they had a CS student in the team who has learned about EM in a data science class.

So we decided to get involved. Our team, which we will call the EM team, consisted of Professor AnHai Doan, a Ph.D. student in CS, and an hourly student in CS. Our team took over the matching work. We viewed the UMETRICS team as domain experts, and consulted with them in that capacity (by meeting for an hour per week and via emails). Our goal here is to simulate and understand the collaboration between an EM team and a domain expert team, as such settings commonly occur in real-world matching projects. Henceforth "we" refers to the EM team (in CS), when there is no ambiguity.

We decided to use PyMatcher to perform EM, by following its how-to guide (see the PyMatcher's homepage for this guide). In the following sections, we discuss the steps we followed. We intentionally discuss them in details, to show all the steps that human users must perform. We also discuss them in chronological order to show how zig-zag the process was. Finally, we discuss details such as where the files were stored, how the two teams communicated, to highlight the logistic aspects of executing such an EM project in a distributed fashion.

## 4 UNDERSTANDING THE DATA

We received the raw data from the UMETRICS team in a Google Drive folder. We started by exploring to understand the tables in the datasets, specifically to understand the "entities" in these tables and the relationships among the entities.

We first opened the raw data and found six CSV tables with the "UMETRICS" prefix and one CSV table with the "USDA" prefix. From the table names, we assumed that the names with the "UMETRICS" prefix correspond to UMETRICS-related tables and the name with the "USDA" prefix corresponds to the USDA table.

Next, we explored each table to obtain a brief understanding of the information included in it and the data values of its columns. Specifically, we browsed a few sample rows that were randomly selected from the table and examined general statistics such as the number of unique values, number of missing values, mean, median, etc., for each column.

UMETRICS

| Table Name | Num. Rows | Num. Cols |
|---|---|---|
| UMETRICSAwardAggMatching | 1336 | 13 |
| UMETRICSEmployeesMatching | 1454070 | 13 |
| UMETRICSObjectCodesMatching | 4574 | 3 |
| UMETRICSOrgUnitMatching | 264 | 5 |
| UMETRICSSubAwardMatching | 21470 | 23 |
| UMETRICSVendorMatching | 377746 | 21 |

USDA

| Table Name | Num. Rows | Num. Cols |
|---|---|---|
| USDAAwardMatching | 1915 | 78 |

**Figure 2: Summary of the original UMETRICS and USDA tables given by the UMETRICS team.**

Figure 2 lists the number of rows and columns for each table. Figure 3 shows a few rows from UMETRICS tables, and Figure 4 shows a few rows from the USDA table.

To explore the smaller tables (which have fewer than 300K tuples), we used pandas [25] and MS Excel. To explore the larger tables, we used SQLite. To profile the tables, we used the pandas-profiling tool [1] and custom Python scripts.

**The UMETRICS Tables:** The schemas of the six UMETRICS tables are as follows:

```
UMETRICSAwardAggMatching(UniqueAwardNumber,AwardTitle,
FundingSource,FirstTransDate, LastTransDate,
RecipientAccountNumber,TotalOverheadCharged,
TotalExpenditures,NumberOfTransactions,
DataFileYearEarliest, DataFileYearLatest,
SubOrgUnit, CampusID)

UMETRICSSubAwardMatching(UniqueAwardNumber, Address,
BldgName, City, Country, DUNS, DomesticZipCode, EIN,
ForeignZipCode, ObjectCode, OrgName, OrganizationID,
POBox, PeriodEndDate, PeriodStartDate, RecipientAccountNumber,
SrtName, SrtNumber, State, StrName, StrNumber,
SubAwardPaymentAmount, DataFileYear)

UMETRICSOrgUnitsMatching(CampusId, SubOrgUnit, CampusName,
SubOrgUnitName, DataFileYear)

UMETRICSEmployeeMatching(UniqueAwardNumber, PeriodStartDate,
PeriodEndDate, RecipientAccountNumber,
DeidentifiedEmployeeIdNumber, FullName,
OccupationalClassification, JobTitle, ObjectCode,
SOCCode, FteStatus, ProportionOfEarningsAllocated,
DataFileYear)

UMETRICSVendorMatching(UniqueAwardNumber, PeriodStartDate,
PeriodEndDate, RecipientAccountNumber, ObjectCode,
OrganizationID, EIN, DUNS, VendorPaymentAmount,
OrgName, POBox, BldgNum, StrNumber, StrName, Address,
City, State, DomesticZipCode, ForeignZipCode, Country,
DataFileYear)

UMETRICSObjectCodesMatching(ObjectCode, ObjectCodeText,
DataFileYear)
```

**Entities of the UMETRICS Tables:** Based on our exploration and profiling of the UMETRICS tables, we inferred the entities of each table as much as we could. "UMETRICSAwardAggMatching" included information about awards, i.e., research grants. It

**UMETRICSSubAwardMatching**

| Unique Award Number | Address | BldgNum | City | Country | DUNS | Domestic ZipCode | EIN | Foreign Zip Code | Object Code | Org Name | Organization ID | POBox | Period End Date | Period Start Date | Recipient Account Number | Srt Name | Srt Number | State | Str Name | Str Number | SubAward Payment Amount | Data File Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 93.395 MSN1006 83 | 8701 Watertown Plank Road | NaN | Milwaukee | US | 937639060 | 53226-05 09 | 390806261 | NaN | 3845 | MEDICAL COLLEGE OF WISCONSIN | 007H055 | NaN | 2008-02-21 | 2008-02-21 | MSN100683 | NaN | NaN | WI | NaN | NaN | 4114 | 2008 |
| 47.078 MSN1021 20 | 110 TECHNOLOGY CENTER BLDG | NaN | UNIVERSI TY PARK | US | 3403953 | 16802-70 00 | 246000376 | NaN | 3845 | PENNSYLVANIA STATE UNIVERSITY | G067944 | NaN | 2007-11-05 | 2007-11-05 | MSN102120 | NaN | NaN | PA | NaN | NaN | 2843.97 | 2008 |

**UMETRICSAwardAggMatching**

| UniqueAward Number | AwardTitle | Funding Source | FirstTrans Date | LastTrans DateDate | Recipient Account Number | Total Overhead Charged | Total Expenditures | NumberOf Transactions | DataFile YearEarliest | DataFile YearLatest | SubOrg Unit | Campus ID |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00.070 03CS-11231300 -031 | MAPPING THE 1990 WUI AND 1990-2000 WUI CHNAGE AT THE CENSUS BLOCK LEVEL ACROSS THE UNITED STATES | USDA, FOREST SERVICE | 2007-07-01 | 2008-07-31 | MSN106198 | 0 | 1296.43 | 5 | 2008 | 2009 | 7 | UWMSN |
| 10.203 WIS01717 | Pathogen and host variability in alfalfa with regard to Aphanomyces root rot | HATCH ACT FORMULA FUND | 2014-01-01 | 2016-06-30 | MSN158789 | 0 | 81171.5 | 30 | 2014 | 2016 | 7 | UWMSN |

**UMETRICSObjectCodesMatching**

| ObjectCode | ObjectCode Text | DatFile Year |
|---|---|---|
| 1000 | Salary Default | 2008 |
| 4880 | Domest Govt’s Documents | 2008 |

**UMETRICSEmployeeMatching**

| UniqueAward Number | PeriodStart Date | PeriodEnd Date | Recipient Account Number | Deidentified EmployeeId Number | FullName | Occupaitional Classification | JobTitle | Object Code | SOC Code | Fte Status | ProportionOf Earnings Allocated | Data File Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 81.049 DE-FG02-95ER40896 | 2007-07-01 | 2007-07-31 | MSN10353 4 | B6D6B9B8C09 CC8993C7A7E EDFA7097052 3B2A8BC | CARLSMITH, DUNCAN L | Faculty | Professor | 1003 | 25-1000.0 | 1 | 1 | 2008 |
| 00.600 MSN108110 | 2007-10-01 | 2007-10-31 | MSN10811 0 | 37CCA6327EE C785343CE2A F84C02B72553 4F2127 | FIORE, MICHAEL C | Faculty | Professor | 1001 | 25-1000.0 | 1 | 0.05 | 2008 |

**UMETRICSOrgUnitMatching**

| CampusID | SubOrgUnit | CampusName | SubOrg Unit Name | DatFile Year |
|---|---|---|---|---|
| UWMSN | 37 | University of Wisconsin - Madison | Wisconsin Collaboratory for Enhanced Learning | 2011 |
| UWMSN | 85 | University of Wisconsin - Madison | University Housing | 2008 |

**UMETRICSVendorMatching**

| Unique Award Number | PeriodStart Date | PeriodEnd Date | Recipient Account Number | Object Code | Organization ID | EIN | DUNS | Vendor Payment Amount | Org Name | POBox | Bldg Num | Str Number | Str Name | Address | City | State | Domestic ZipCode | Foreign ZipCode | Country | Data File Year |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00.500 MSN1027 93 | 2008-02-20 | 2008-02-20 | MSN102 793 | 3740 | 0000000066-1 | NaN | NaN | 698.8 | CAPITAL NEWSPAPER S INC | NaN | NaN | NaN | NaN | PO BOX 9069 | MADISON | WI | 53708 | NaN | US | 2008 |
| 81.049 DE-FG02-95E R40896 | 2007-07-13 | 2007-07-13 | MSN103 534 | 3105 | 0000000066-1 | NaN | NaN | 49.58 | ACE HARDWARE CENTER | NaN | NaN | NaN | NaN | 1398 WILLIA MSON ST | MADISON | WI | 53703 | NaN | US | 2008 |

**Figure 3: Example rows from the UMETRICS tables.**

| Accession Number | Project Title | Sponsoring Agency | Funding Mechanism | Award Number | Initial Award Fiscal Year | Recipient Organization | Recipient DUNS | Project Director | Multistate Project Number | Project Number | Project Start Date | Project End Date | Project Start Fiscal Year | … | … | Financial: USDA Contracts, Grants, Coop Agmt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 175763 | GENETIC ORGANIZATION AND EPIGENETIC SILENCING OF MAIZE R GENES | State Agricultural Experiment Station | State Funding | NaN | NaN | SAES – UNIVERSITY OF WISCONSIN | NaN | Kermicle, J.L | NaN | WIS04059 | 1997-07-01 | 2010-09-30 | 1997 | … | … | NaN |
| 190977 | The Changing Location and Extent of the Wildland-Urban Interface During the 1990's | State Agricultural Experiment Station | State Funding | NaN | NaN | SAES – UNIVERSITY OF WISCONSIN | NaN | Hammer, R | NaN | WIS04593 | 2001-10-01 | 2011-09-30 | 2002 | … | … | NaN |

**Figure 4: Example rows from the USDA table.**

included funding information for the research projects in the university. “UMETRICSSubAwardMatching” included information about how the awards are split into multiple sub-awards to fund the research projects.

“UMETRICSOrgUnitsMatching” included information about different organization units to which the awards were given. “UMETRICSEmployeeMatching” included information about the employees in the university, and “UMETRICSVendorMatching” included information about the vendors interacting with the university. We were not clear about the information included in the “UMETRICSObjectCodesMatching” table.

**Relationships among the UMETRICS Tables:** We observed that “UMETRICSAwardAggMatching” was the central table to which most other tables (except “UMETRICSObjectCodesMatching”) could be joined using a key-foreign key relationship. For this table (i.e., “UMETRICSAwardAggMatching”), the attribute “UniqueAwardNumber” was the primary key.

We found that the tables “UMETRICSEmployeeMatching,” “UMETRICSVendorMatching,” and “UMETRICSSubAwardMatching” included the “UniqueAwardNumber” column, which could be joined with the ”UniqueAwardNumber” of the “UMETRICSAwardAggMatching” table. The table “UMETRICSOrgUnitsMatching” included a “CampusId” column that could be joined with “CampusId” in the “UMETRICSAwardAggMatching” table. The table “UMETRICSObjectCodesMatching” included an “ObjectCode” column that could be joined with the “ObjectCode” column in the “UMETRICSEmployeeMatching” table.

**The USDA Table:** Only one table contained USDA information (see Figure 4). This table has 78 columns. Because of space constraints, a partial schema of this table including a few columns is shown below.

```
USDAAwardMatching(AccessionNumber, ProjectTitle,
SponsoringAgency, FundingMechanism, AwardNumber, ...,
ProjectNumber, ProjectStartDate, ProjectEndDate, ...,
ProjectDirector,  ...,
Financial: USDAContracts, Grants, Coop Agmt)
```

## 5 UNDERSTANDING MATCH DEFINITION

After exploring the tables, we have obtained an understanding of the entities and their relationships. However, we did not know how to use these tables to match the awards (i.e., grants), e.g., which tables are relevant for matching? What does it mean to be a match between the UMETRICS and USDA datasets?

In response, the UMETRICS team sent us a matching document, which discusses the most relevant tables, provides a matching definition, and provides a few sample matching and non-matching record pairs. This document stated that only three tables (among seven) were most relevant for matching and provided the following matching definition.

- ($M_1$) If a part of "UniqueAwardNumber" in UMETRICS matches "Award Number" in USDA, then the record pair can be considered a match. Specifically, "UniqueAwardNumber" can take the form "XX.XXX YYYY-YYYY-YYYYY-YYYYY". Thus if "YYYY-YYYY-YYYYY-YYYYY" matches the "Award Number," then the record pair is a match. An example of such a match is shown in Figure 5.
- ($M_2$) A number of records in USDA do not have values for "Award Number." In such cases, the records may be matched by checking whether the "AwardTitle" in UMETRICS and the "Project Title" in USDA are similar. An example of such a match is shown in Figure 6.
- ($M_3$) A record pair from UMETRICS and USDA can also be matched by comparing the individuals involved in the project.

From the above matching definition, we could infer one positive matching rule based on $M_1$. Specifically, for a record pair from UMETRICS and USDA tables, if the second part of "UniqueAwardNumber" in the UMETRICS record matches exactly the "Award Number" in the USDA record, then the record pair could be declared a match.

It is possible to use this positive rule to filter out all the positive matches, then proceed with a smaller set for matching. However, we did not do that because we were not sure if the match definition had been stabilized as yet. Instead, we decided to incorporate this rule as a part of the blocking step (as we will see soon).

Apart from $M_1$, the other two instructions ($M_2$ and $M_3$) in the above matching definition are not precise. For example, in $M_2$, what does it mean to say "similar"? Further, if the award and project titles match exactly, but are very generic (e.g., "Lab Supplies"), then we still cannot conclude that the records match. So we cannot capture these matching instructions as rules, to be applied to the tables to obtain the matches.

As this example suggests, matching definitions are often written in English, in a verbose and imprecise fashion. Business owners often train analysts to perform matching. These analysts gain experience over time and tune their understanding of a "match." They do this by exploring a wide variety of examples and checking with the business owners when in doubt. *This suggests that understanding a matching definition is an iterative process that involves continuous interaction with the business owners.*

## 6 PRE-PROCESSING THE DATA

Recall that we were given six UMETRICS tables (describing how the funding from different agencies was used for the research projects at UW-Madison) and one USDA table (describing how the funding from USDA was used for research projects at UW-Madison).

| UMETRICS | | USDA | |
|---|---|---|---|
| *field* | *value* | *field* | *value* |
| UniqueAwardNumber | 10.200 2008-34103-19449 | Accession Number | 214335 |
| AwardTitle | DEVELOPMENT OF IPM-BASED CORN FUNGICIDE GUIDELINES FOR THE NORTH CENTRAL STATES | Project Title | Development of IPM-Based Corn Fungicide Guidelines for the North Central States |
| FirstTransDate | 10/1/08 | Award Number | 2008-34103-19449 |
| FirstTransDate | 10/1/08 | Project Start Date | 8/15/08 |
| | | Project End Date | 8/14/11 |
| | | Project Director | ESKER, PAUL |

**Figure 5: A matching pair based on Award Number.**

| UMETRICS | | USDA | |
|---|---|---|---|
| *field* | *value* | *field* | *value* |
| UniqueAwardNumber | 10.203 WIS01040 | Accession Number | 206746 |
| AwardTitle | SWAMP DODDER (CUSCUTA GRONOVII) APPLIED ECOLOGY AND MANAGEMENT IN CARROT PRODUCTION | Project Title | Swamp Dodder (Cuscuta gronovii) Applied Ecology and Management in Carrot Production |
| FirstTransDate | 10/1/07 | Award Number | - |
| LastTransDate | 12/31/08 | Project Start Date | 10/1/06 |
| | | Project End Date | 9/30/08 |
| | | Project Director | Colquhoun, J. |

**Figure 6: A matching pair based on Award Title.**

Even with this moderate number of tables, the matching process will be difficult if we have to consider all of them. So we decided to *subset the tables* (i.e., selecting only a portion of information from the tables, perhaps even using just a subset of the tables) and *apply transformations* to obtain only two tables that can be used for matching. To do so, we used the matching document provided by the UMETRICS team.

Specifically, we created two tables "UMERICSProjected" and "USDAProjected" (later we will match these two tables), using the following steps.

**(1)** First, from the six UMETRICS tables, we selected two tables judged most relevant for matching by the UMETRICS team (as discussed in the matching document): "UMETRICSAwardAggMatching" and "UMETRICSEmployeeMatching". (Naturally we also keep the table "USDAAwardMatching".)

**(2)** Next, we checked that "UniqueAwardNumber" and "Accession Number" were indeed the key columns in the "UMETRICSAwardAggMatching" and "USDAAwardMatching" tables, respectively. We also checked that "UniqueAwardNumber" was indeed a foreign key in the "UMETRICSEmployeesMatching" table with valid values, and could be joined with the "UMETRICSAggAwardMatching" table. We used PyMatcher and pandas to perform these validations.

**(3)** Then we checked the four remaining UMETRICS tables to see if they contained any information useful for matching. To do this, we manually examined the names of the attributes of these four UMETRICS tables and compared these names to the names of the attributes of the USDA table, to find attribute pairs with similar names. We found that "Recipient Organization" and

"Recipient DUNS" from the USDA table were similar to "OrgName" and "DUNS" in the "UMETRICSVendorMatching" table[1].

Next, we checked if the attributes with similar names have similar values. Specifically, we checked for any overlap of values and compared the distributions of values using mean, median, etc. We did this using pandas and custom Python scripts. We found that the values of "OrgName"and "DUNS" from the "UMETRICSVendorMatching" table did not overlap with the values of "Recipient Organization" and "Recipient DUNS".

We concluded that the four remaining UMETRICS tables do not share any information with the USDA table, and thus are not useful for matching. Thus, we ignore them in the subsequent pre-processing steps.

**(4)** Finally, we applied transformations to the selected tables. Specifically, we (a) projected out the UMETRICS and USDA tables to create two tables with relevant columns for matching, (b) matched the columns between the tables and renamed them with the same names, and (c) added an ID column to each table. We now elaborate on these steps.

**(4.a)** First, we projected and kept from the two tables "UMETRICSAwardAggMatching" and "USDAAwardMatching" only attributes that are relevant for matching (we consulted the matching document and the UMETRICS team about which attributes to keep). This produced the following two tables:

```
UMETRICSProjected(UniqueAwardNumber, AwardTitle,
FirstTransDate, LastTransDate)

USDAProjected(AwardNumber, ProjectTitle, ProjectStartDate,
ProjectEndDate, AccessionNumber, ProjectDirector)
```

The "AccessionNumber" was included in the "USDAProjected" table because the UMETRICS team required the output matches to be listed as pairs of "UniqueAwardNumber" and "AccessionNumber."

**(4.b)** Next, we matched the column names between the two tables and renamed them with the same name. Specifically, we matched "UniqueAwardNumber," "AwardTitle," "FirstTransDate," "LastTransDate" from the "UMETRICSProjected" table to "AwardNumber," "ProjectTitle," "ProjectStartDate," "ProjectEndDate," respectively. We named them "AwardNumber," "AwardTitle," "FirstTransDate," "LastTransDate." We renamed "ProjectDirector" in the "USDAProjected" table as "EmployeeName." The updated schemas are shown below.

```
UMETRICSProjected(AwardNumber, AwardTitle,
FirstTransDate, LastTransDate)

USDAProjected(AwardNumber, AwardTitle, FirstTransDate,
LastTransDate, AccessionNumber, EmployeeName)
```

Then we added a new column, "EmployeeName," to the "UMETRICSProjected" table. To do so, we joined this table with the "UMETRICSEmployeesMatching" table on the "AwardNumber" and "UniqueAwardNumber" columns. There were multiple employee names for the same award in the "UMETRICSEmployeesMatching" table. Therefore, for each award, these employee names were concatenated, and each employee name was separated by the | character.

**(4.c)** Finally, we added an ID column ("RecordId") to both the "UMETRICSProjected" and "USDAProjected" tables, to uniquely identify each record in each table. The final schema of the two tables are (see Figure 7):

[1]What we did here is essentially schema matching [27]. But we did it manually due to the relatively small sizes of the tables.

**UMETRICSProjected**

| RecordId | AwardNumber | AwardTitle | FirstTrans Date | LastTrans Date | Employee Name |
|---|---|---|---|---|---|
| 78 | 00.070 58-3655-8-123 | FY08 RSA TASK ORDER AGREEMENT 123 | 2007-10-01 | 2009-05-31 | NaN |
| 198 | 10.000 14-JV-11242309-078 | Fire Risk and Fire Mitigation Zones and the WUI | 2014-09-01 | 2016-02-29 | HELMERS\| DAVID P\| STEWART\| SUSAN I |

**USDAProjected**

| RecordId | Award Number | AwardTitle | FirstTransDate | LastTrans Date | Employee Name | Accession Number |
|---|---|---|---|---|---|---|
| 63 | NaN | Food Research | 2000-07-01 | 2011-09-30 | Yu J | 186338 |
| 172 | NaN | Potato Research | 1998-10-01 | 2011-09-30 | Kelling, Keith | 181962 |

**Figure 7: Sample rows of the UMETRICSProjected and USDAProjected tables.**

```
UMETRICSProjected(RecordId, AwardNumber, AwardTitle,
FirstTransDate, LastTransDate, EmployeeName)

USDAProjected(RecordId, AwardNumber, AwardTitle,
FirstTransDate, LastTransDate, AccessionNumber,
EmployeeName)
```

We used pandas, PyMatcher, and custom Python scripts to perform the join and other transformations.

## 7 BLOCKING

After applying the transformations, the resulting tables, "UMETRICSProjected" and "USDAProjected", have just 1336 and 1915 records, respectively.

Since they are small, can we just match all pairs of records in their Catersian product? In other words, is blocking necessary? It turns out that blocking is still required even in this case. This is because to perform learning-based matching and to evaluate the match results, we must first take a sample from this set (of record pairs in the Cartesian product) and label them.

In our case, however, the Cartesian product of the input tables has 2.5M record pairs, and most of them would be non-matches. Random sampling from this set will result in very few matches. Therefore, we still have to perform blocking to remove obvious non-matching record pairs, so that later when we sample we can obtain more matches in the samples.

We used the matching definition provided by the UMETRICS team to guide the blocking step. We proceeded as follows:

**(1)** First, we applied a blocking scheme to include all record pairs that satisfy $M_1$. This is because if $M_1$ is indeed a positive matching rule, then all record pairs satisfying $M_1$ must be included in the candidate set (to be fed into the matching step).

Recall that $M_1$ declares a record pair a match if the second half of the "AwardNumber" attribute of the "UMETRICSProjected" table matches exactly the "AwardNumber" attribute of the "USDAProjected" table. To find all record pairs that satisfy $M_1$, we applied an attribute equivalence (AE) blocker to these tables. This blocker includes a record pair (in the candidate set) only if the blocking attributes of both input tables agree.

In our case, the AE blocker cannot be applied directly because the "AwardNumber" from "UMETRICSProjected" and "USDAProjected" cannot be compared for an exact match. So we first used a regular expression to extract the suffix of "AwardNumber" of the "UMETRICSProjected" table and stored the result as a temporary column, "TempAwardNumber," in the same table. Then we applied the AE blocker using "TempAwardNumber" from the

"UMETRICSProjected" table and "AwardNumber" from the "USDAProjected" table as blocking attributes. Finally, we removed the temporary column ("TempAwardNumber") from the "UMETRICSProjected" table. This blocking scheme produced a candidate set of record pairs $C_1$.

**(2)** Next, based on the matching definition $M_2$, we decided to include the record pairs that have similar award titles. We examined a sample of the award titles in the "USDAProjected" and "UMETRICSProjected" tables, and observed that the titles often have multiple tokens (i.e., words in this case).

Intuitively, two similar award titles should share at least a few tokens. So we applied an overlap blocker to the input tables using "AwardTitle" as the blocking attribute. This blocker discards a record pair if the number of shared tokens (in the blocking attribute) is less than an overlap threshold $K$.

Specifically, we normalized all the strings in the "AwardTitle" column by lower casing and removing special characters (e.g., single/double quotation marks, hash symbols, exclamation marks, round/curly braces, etc.). Then we performed overlap blocking using a word-level tokenizer, using the overlap threshold 3 after trying a few other thresholds (e.g., the threshold of 1 resulted in 200K record pairs, and a threshold of 7 resulted in a few hundred record pairs). This produced a candidate set $C_2$.

**(3)** The overlap blocker drops a record pair if the number of tokens in the blocking attribute was less than the overlap threshold $K$ (in our case, $K$ was 3). So we examined the award titles between the two tables to see if similar titles with fewer than 3 tokens exist, and we found quite a few such title pairs[2].

To include these record pairs, we applied an overlap-coefficient blocker, using "AwardTitle" as the blocking attribute. For any two strings $X$ and $Y$ we have $overlap\_coefficient(X, Y) = |X \cap Y| / \min(|X|, |Y|)$ (assuming $X$ and $Y$ have been tokenized into two sets). This blocker is similar semantics-wise to the overlap blocker. However, it returns a score between 0 and 1, regardless of the title length, and thus can handle the case where a title has fewer than 3 tokens. To apply this blocker, we first lower cased all the strings in the "AwardTitle" column, removed special characters, then performed overlap-coefficient blocking using a word-level tokenizer and a threshold of 0.7 (after trying a few other thresholds). This produced a candidate set $C_3$.

**(4)** Next, we unioned $C_1$, $C_2$, and $C_3$ to obtain a consolidated candidate set $C$, which has 3177 record pairs[3].

We then checked for any potentially missing matches in $C$ using the blocking debugger of PyMatcher [23]. Briefly, this debugger takes the two input tables ("UMETRICSProjected" and "USDAProjected") and the candidate set $C$, and returns the list of record pairs that (a) are in the Cartesian product of the two tables but not in $C$, and (b) are judged to be potential matches by the debugger. These pairs are ranked in decreasing likelihood of being matches. If the user does not see many true matches in this list (e.g., by manually examining the top 100 pairs), then he/she can conclude that the blocking process probably has not killed off many true matches. See [23] for details, including how the debugger performs the above process fast.

In our case, we observed that the top record pairs returned by the blocking debugger were not matches, and hence we decided to stop modifying the blocking pipeline. We used PyMatcher for blocking, and used custom Python scripts and pandas commands to preprocess the columns before applying blocking[4].

## 8 SAMPLING AND LABELING

After blocking, we wanted to obtain and label a sample of record pairs from the candidate set $C$. (Later we need this labeled sample to select the best learning-based matcher and then train this matcher to predict matches in the candidate set $C$.)

Sampling and labeling is not straightforward because the candidate set $C$ has relatively few matches and the match definition is still evolving. Further, since we had to ask the UMETRICS team to label, we had to manage the logistics for labeling.

**Setting Up:** We first emailed the UMETRICS team, then followed up with a face-to-face meeting to discuss the logistics. Initially, we proposed to email the record pairs as a CSV file that they could download and use tools such as MS Excel to label pairs as a match or a non-match. The UMETRICS team wanted a tool with better UI that multiple team members could use. So we developed a simple cloud-based labeling tool with a good UI, but the tool was limited in that only one person could label at any time. The UMETRICS team accepted this and said they would discuss the matter internally and schedule the labeling.

**Sampling and Labeling:** Next, we performed sampling and labeling iteratively. Our goal was to obtain a sufficient number of positive matches in the labeled set. We first took a random sample of 100 record pairs from the candidate set $C$, uploaded the sampled pairs into the cloud-based labeling tool, then asked the UMETRICS team to label the record pairs as "Yes", "No", or "Unsure"[5].

The UMETRICS team trained a student to label using the tool. Meanwhile, we labeled the same set of record pairs, using our own understanding of the match definition. Afterward we cross-checked their labels against ours and observed 22 mismatched labels. Specifically, one record pair was labeled a non-match despite satisfying the matching rule $M_1$. The other 21 pairs had similar award titles, but labeled as a mix of match, non-match, and primarily unsures.

In a face-to-face meeting, the UMETRICS team confirmed that the record pair satisfying $M_1$ must be declared as match (they also confirmed that $M_1$ is indeed a positive matching rule). For others, they mentioned that, though the award titles were similar, some of them were not unique enough to be declared matches.

They said that they would have a closer look at the mismatches. After the discussion, we shared the record pairs with mismatched

[2]Specifically, we first used PyMatcher to find all title pairs whose Jaccard score (over a 3-gram tokenization) exceeds a threshold. Next, we kept only those pairs where at least one title has fewer than 3 words. Finally, we took a random sample of these pairs and manually examined the sample.

[3]It turned out that we want both $C_2$ and $C_3$ in the consolidated candidate set. $C_2$ and $C_3$ have 2,937 and 1,375 record pairs, respectively. $|C_2 \cap C_3|$ = 1,140, $|C_2 - C_3|$ = 1,797, and $|C_3 - C_2|$ = 235. Our examination revealed that if two titles are similar but share few tokens, the overlap blocker will include the pair, but the coefficient blocker will discard it. So we cannot just use $C_3$ and discard $C_2$.

[4]PyMatcher's blocking methods use string filtering techniques where appropriate to speed up the blocking process.

[5]It turned out that for many real-world datasets that we have seen, even domain experts had troubles labeling certain pairs, due to dirty, incomplete, or cryptic data. Hence the option "Unsure". As we will see, we ignore "Unsure" pairs in training and evaluating learning-based matchers. Our reasoning is that if even domain experts cannot label these pairs, it is unfair to ask learning algorithms to handle them. And if we can ask learning algorithms to label these pairs, how can we evaluate their accuracy, given that even the domain experts cannot label them?

We also believe that if the number of "Unsure" pairs is too high, that indicates that the dataset is too dirty, incomplete, or cryptic, and the domain experts should clean the dataset before EM is attempted. In this case, as we will see, 300 pairs were labeled for selecting/training matchers, out of which 32 were labeled "Unsure". Later 400 pairs were labeled for evaluating matchers, out of which 16 were labeled "Unsure". The UMETRICS team said they were okay with the quality of this dataset (i.e., judging the EM result from this dataset to be still useful for their domain science research).

labels using Google Sheets. The UMETRICS team updated 4 labels to "Yes" (keeping the labels of the remaining pairs). After the updated labels were submitted, 15 pairs were labeled as "Yes", 66 labeled "No" and 19 labeled "Unsure".

Next, because we had only 15 positive matches, we decided to obtain more labeled pairs. We discussed (via email) and asked the UMETRICS team to label at least 100-200 more record pairs to obtain a sufficient number of positive matches. Following this, we internally decided to obtain labeled record pairs in two iterations, with 100 record pairs per iteration[6]. The first iteration produced 29 "Yes" pairs, 64 "No", and 7 "Unsure". The second iteration produced 24 "Yes", 72 "No", and 4 "Unsure".

We now have 300 labeled pairs, consisting of 68 "Yes", 202 "No", and 30 "Unsure". Because we have obtained a sufficient number of positive matches, we stopped seeking more labeled pairs.

**Debugging the Labeled Sample:** Next, since the labeled pairs will be used to select and train a matcher, any labeling errors can impact the predicted matches. To minimize this impact, we decided to debug the labels.

To do so, we used leave-one-out cross-validation: we trained an ML matcher on all labeled record pairs except one, applied the matcher to predict the label for the left-out record pair, compared this predicted label with the label given by the UMETRICS team, and repeated the process. We used random forest as the ML matcher, and removed the unsure and sure matches (record pairs that satisfy $M_1$) from the labeled data before debugging[7]. We observed the following discrepancies:

($D_1$) Record pairs were predicted matches, but labeled non-matches if the award titles were very similar but the award title in the "USDAProjected" table included the suffix "NC/NRSP."

($D_2$) Record pairs were predicted matches but labeled non-matches if the award numbers were different but the award titles were the same or very similar.

($D_3$) Record pairs were predicted matches but labeled a mix of matches and non-matches if the award number was missing from "USDAProjected" but the award titles were very similar.

We shared example record pairs (using Google Sheets) for each of the above discrepancies, and discussed via email and face-to-face meetings with the UMETRICS team. This team (based on their domain knowledge) decided that all labels in $D_1$ should be updated as unsure (even they did not know if these were matches). All labels in $D_2$ should be retained. For $D_3$, the labels must be updated as matches if the transaction dates for the awards are within a difference of few years (e.g., two years). This produced a final set of 300 labeled pairs, consisting of 68 "Yes", 200 "No", and 32 "Unsure."

## 9 MATCHING

We now use the above set of 300 labeled record pairs to find matches between the "UMETRICSProjected" and "USDAProjected" tables. Specfically, we used this labeled set to select a good learning-based matcher, then applied this matcher to the pairs in the candidate set $C$ (obtained after blocking) to predict matches.

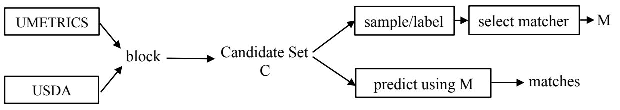

**Figure 8: The initial EM workflow for matching the UMETRICS and USDA datasets.**

**Selecting a Good Matcher:** To do this, we used PyMatcher. Let $G$ be the set of 300 labeled pairs. First, we removed the pairs labeled "Unsure" and sure matches (i.e., pairs satisfying $M_1$) from $G$, then converted $G$ into a set of feature vectors $H$ (each pair was converted into a feature vector; we used PyMatcher to automatically generate a set of features $F$, then used $F$ to generate feature vectors).

PyMatcher uses the scikit-learn package, and learning methods in this package cannot work with missing values in the feature vectors. So in the next step we filled in the missing values in the feature vectors in $H$ (with the mean values of the respective columns).

Next, we selected the best (i.e., the most accurate) matcher using five-fold cross validation on $H$. Among decision tree, SVM, random forest, logistic regression, naive Bayes, and linear regression matchers (supplied by PyMatcher; these matchers in turn use corresponding learning methods in scikit-learn), we found the random forest (RF) matcher had the highest $F_1$ accuracy (averaged over the five folds).

Next, we debugged the RF matcher to try to improve its accuracy. Again, we used the debugging method described in PyMatcher. This method tried to find the mismatches, i.e., those pairs where the RF matcher predicted incorrectly, then examine these mismatches and take actions to fix them (see [17]). Toward this goal, we randomly split $H$ into two sets $I$ and $J$, trained the RF matcher on $I$, then applied it to $J$ and identified mismatches in $J$, i.e., pairs in $J$ where the label given by the RF matcher differs from the label given in the sampling and labeling process described earlier. We then trained the RF matcher on $J$ and applied it to $I$, to identify the mismatches in $I$.

It turned out many mismatches occurred due to award titles having different letter cases. So we added more features to handle this problem[8]. We then performed cross validation to select the best matcher again. Now the decision tree performed the best with 97% precision, 95% recall, and 94.7% $F_1$, on average. We debugged this matcher using the decision tree matcher debugger of PyMatcher, but were not able to improve accuracy. So we stopped and selected this matcher as the best matcher.

**Applying the Selected Matcher:** We first trained the decision tree matcher $M$ on the set of feature vectors $H$ (i.e., the set of 300 labeled record pairs). Next, we removed the record pairs satisfying the positive matching rule $M_1$ from the candidate set $C$ (which was obtained after blocking). There were 210 record pairs that satisfy $M_1$. Next, we converted the revised set $C$ into a set of feature vectors $C'$ (each pair becomes a feature vector) and imputed the missing values with the mean of the respective columns. Finally, we applied the trained matcher $M$ to $C'$. This produced 807 matches. Figure 8 shows the overall EM workflow.

Counting also the record pairs that satisfy the positive matching rule, we obtained a total of 1,017 matches. We shared these matches in a CSV file with the UMETRICS team and discussed them in a face-to-face meeting.

[6]Since labeling is time consuming, we wanted to do this in iterations (rather than asking the UMETRICS team to label all 200 pairs in one shot) to make sure we can catch and handle any possible labeling glitches early.

[7]For cross validation, we need to convert each record pair into a feature vector. To do this, we applied PyMatcher to the schemas of the two tables "UMETRICSProjected" and "USDAProjected" to automatically generate a large set of features, which include both string related features (e.g., Jaccard over 3grams, edit distance, etc.) and numeric features (e.g., absolute difference, exact match, etc.). See the PyMatcher homepage for a description of how these features are automatically generated. We then used these features to convert each record pair into a feature vector.

[8]We did not not lowercase all words in the pre-processing step, because our experience with many EM projects suggests that that often resulted in a loss of information. So we only lowercased where necessary.

## 10 HANDLING MANY COMPLICATIONS

What happened next were many complications regarding whether the current match definition (i.e., what it means to be a match) is incorrect and should be revised, and how to acommodate more data (for the tables), which was accidentally omitted earlier.

**Should We Match at the Cluster Level?** During the discussion with the UMETRICS team, we observed a gap between their definition of a match between the UMETRICS and USDA awards and our understanding of the same. Specifically, in our set of 1,017 matches, there are many one-to-many matches, e.g., a record in the "UMETRICSProjected" table matches many records in the "USDAProjected" table. The UMETRICS team insisted that matches should be one-to-one, i.e., a record in "UMETRICSProjected" should match at most one record in "USDAProjected".

We were confused by this requirement. Recall from Section 3 that a grant given to a project may be distributed to many smaller projects (e.g., one per an academic year, or one per CS and one per biology) in the same university, and this information will be recorded in multiple entries. For instance, multiple records in "USDAProjected" can contain information about the exact same award, but for different academic years (e.g., a research group may receive a 3-year award; it can send 3 annual reports to USDA, resulting in 3 records in USDA about this award). Given this, a record in "UMETRICSProjected" can match many records in "USDAProjected" and vice versa.

Upon further discussion, it turned out that what the UMETRICS team really wanted is this. As mentioned earlier, a project may last for many years, and may be associated with many subawards. Each sub-award is captured by a record in a table. So the UMETRICS team wanted to cluster the records in each table, such that all sub-awards go into the same cluster. They then wanted to match these clusters. At this level, insisting that the matches be one-to-one, i.e., a cluster in "UMETRICSProjected" match at most one cluster in "USDAProjected" makes sense.

Unfortunately, since this desire was not conveyed to us earlier (they were not even aware of this sub-award problem originally), we already performed matching at the record level (not at the cluster level), and at this level one-to-many and many-to-one matches make sense.

We then analyzed the one-to-one, one-to-many, and many-to-one match predictions and shared our analysis with the UMETRICS team. Our goal was to show examples of these and their frequency in the set of matches. Tthe reasoning is that if a problem affects only a small number of matches, then it is not worth spending a lot of effort to solve that problem.

We had another discussion with the UMETRICS team. There they decided that the problem does not affect many matches and so probably would have an insignificant effect on their domain science (which relies on these matches). So they opted to keep matching at the record level, and kept the definition that one record in the "UMETRICSProjected" table can match many records in the "USDAProjected" table, and vice versa. This would avoid a redo of the project, i.e., starting by clustering the records, then trying to match the clusters.

**Revising the Match Definition:** Recall that the current UMETRICS repository already has a rule-based matching system, and that our goal is to beat that system. To do so, the UMETRICS team logged into UMETRICS and applied the rule-based matching system to the same data (i.e., the "UMETRICSProjected" and "USDAProjected" tables) to obtain matches. The idea was to compute and compare the precision and recall of the rule-based system to those of our system.

During this process, they discovered that another positive matching rule existed: *"If the award number in UMETRICS matches the project number in USDA, then the record pair is considered a match".* This rule could be used to pull more sure matches directly from the "UMETRICSProjected" and "USDAProjected" tables[9].

Thus, the match definition was revised to include this rule. The question is how would this complicate the EM process. The how-to guide of PyMatcher spells out a well-defined sequence of steps for EM (e.g., blocking, sampling, etc.). We trained the students to follow this sequence. By default, a revised match definition would trigger a re-do of this sequence, which is time consuming. For example, blocking can be easily revised by just using the new positive matching rule to add more record pairs to the candidate set $C$. But sampling and labeling would be time consuming. We need to take a *random* sample of the candidate set $C$, and since $C$ has changed, we need to re-do the random sampling and label any new pair in the sample (that has not been labeled before). *While labeling a small number of pairs seems trivial, in practice it can take days, as we need to upload the pairs to the cloud-based labeling tool, then wait for the* UMETRICS *team, which often cannot label right away, due to other obligations.*

Thus, we did not want to redo the EM process from scratch. To avoid this, first we checked whether the ML matcher was already learning the above positive rule from the labeled data. Specifically, we checked and found that there were 411 pairs in $C$ that satisfy that rule, and out of those 397 were predicted as matches. Thus, our matcher correctly predicted most of the pairs that satisfy the rule as matches. Next, we checked if blocking discarded any pairs that satisfy the new rule. We found that the Cartesian product of the "UMETRICSProjected" and "USDAProjected" tables contain 473 such pairs, but $C$ included only 411. Thus, blocking discarded too many pairs. So the new positive matching rule does have an effect on the EM process, and something must be done.

Our solution was to leave the current EM workflow alone and create a new EM workflow (which will be used together with the current EM workflow). In this new workflow, we wrote a Python script to apply the new matching rule directly to the input tables "UMETRICSProjected" and "USDAProjected" to obtain the sure matches. *This new EM workflow can be viewed as a "patch" of the current EM workflow.* If a pair is predicted by both the old and new workflows, then we take the prediction of the new workflow. *An advantage of this approach is that we did not have to label any new pairs.*

**Handling More Data:** When the UMETRICS team inspected the matches produced by the rule-based matcher at UMETRICS, they found new award numbers that they had not seen before. Further inspection revealed that the original table "UMETRICSAwardAggMatching" was incomplete, missing 496 records.

We received these extra records in a CSV file. Revising the table "UMETRICSAwardAggMatching" (to add these records) and redoing the EM process following the PyMatcher's how-to guide would be time consuming (all the steps, blocking, sampling, labeling, matching, etc. must have been redone). So we followed the same strategy used to handle a change in the match definition. *We keep the current EM workflow "as is" or make only minimal changes to it, then "patch" it with new EM workflows.* This way

[9]"AwardNumber" is already in table "UMETRICSProjected". "ProjectNumber" is not in table "USDAProjected". However, it is in table "USDAAwardMatching" and thus can be easily added to "USDAProjected".

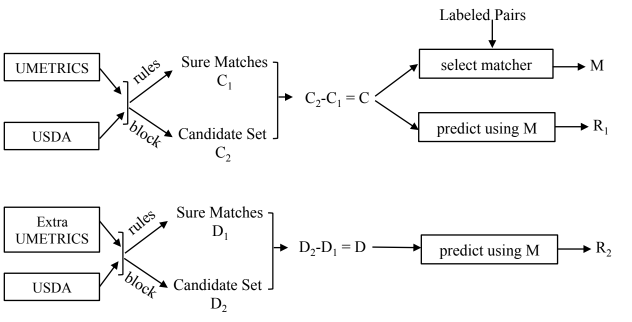

**Figure 9: The updated EM workflow to accommodate extra data and positive matching rules.**

we minimized the changes we had to make to our existing workflow, and minimized the total human effort. This resulted in the following procedure (see Figure 9):

(1) Apply the sure-match rules to the original input tables. Specifically, apply rule $M_1$ (from the match definition) and the positive matching rule involving award number and project number to obtain a set $C_1$.
(2) Apply blocking as before to obtain a candidate set $C_2$.
(3) Remove the sure matches ($C_1$) from $C_2$ to obtain a set $C$; this set $C$ is what will be predicted as matches and non-matches.
(4) Use the labeled set without the sure matches to train the best matcher and predict on $C$, and call the resulting matches $R_1$.
(5) Repeat Steps 1-3 for the extra records in UMETRICS and the whole USDA table until a set of sure matches $D_1$ and a candidate set $D$ are obtained.
(6) Apply the best matcher obtained using labeled data (in Step 4) to $D$ to get a set of match predictions $R_2$.
(7) Return the union of $C_1, D_1, R_1$, and $R_2$ as the set of matches.

The above procedure produced 1,137 matches. Specifically, we first applied the sure-matches rule to obtain 683 sure matches from the original input tables and 55 sure matches from the additional records. Then we applied blocking and removed the sure matches. This resulted in a candidate set from the original input tables having 2,556 record pairs and a candidate set from the extra records having 1,220 record pairs. Next, we removed the sure matches from the labeled set and selected the best matcher, which was a decision tree matcher. Finally, we applied this matcher to the candidate sets and obtained 399 matches from the original tables and no matches from the additional records.

## 11 ESTIMATING ACCURACY

Now that we had finally obtained the matches in our approach (and praying that no more complications arise), we were ready to estimate the accuracies of our solution and the rule-based matcher deployed at UMETRICS, which we will call the IRIS matcher (IRIS is the organization that manages UMETRICS).

Ideally, if we have the true labels for the record pairs in the Cartesian product of the input tables, then we can compute the accuracy, i.e., precision and recall. However, having the true labels would mean that there was no need to do EM in the first place. To address this, we met with the UMETRICS team and decided to follow the accuracy estimation approach described in the Corleone paper [13]: We proceeded as follows:

(1) To use the Corleone approach, both the IRIS matches and our predicted matches *must be from the same candidate set of record pairs*. So we looked for any award number-accession number pairs from the IRIS matches that were not included in our consolidated candidate set (from the original and extra records), which is $E = C_1 \cup C_2 \cup D_1 \cup D_2$ (see Figure 9). We found only one such pair. The UMETRICS team said that the award number in question was a terminated award (no longer valid) and could be discarded safely.

(2) We took a random sample of 200 record pairs from the consolidated candidate set $E$, uploaded it to the cloud-based labeling tool, and asked the UMETRICS team to label. Then we used $E$ to estimate precision and recall, using Formulas 2-3 in Section 6.1 of the Corleone paper [13]. We estimated that our matcher had precision in the range (79.6%, 86.01%) and recall in (96.8%, 99.42%). The IRIS matcher had precision in the range (100%, 100%) and recall in (52.7%, 62.07%). Thus, the IRIS matcher had higher precision, but lower recall, compared to our matcher.

(3) To reduce the large interval sizes of the estimated precision and recall, we asked the UMETRICS team to label another 200 randomly sampled record pairs. Applying the same estimation procedure to all 400 labeled pairs[10], we estimated that our matcher had precision in (75.2%, 80.3%) and recall in (98.1%, 99.6%). The IRIS matcher had precision in (100%, 100%) and recall in (65.1%, 71.8%). The UMETRICS team liked the fact that our matcher was able to find more matches than the IRIS matcher.

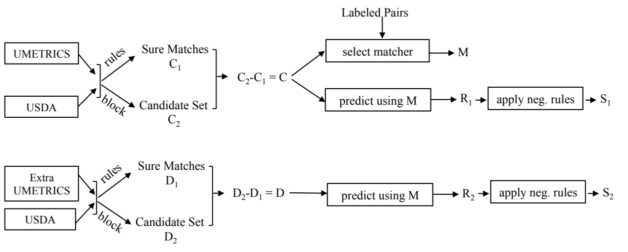

**Figure 10: The final EM workflow with negative rules applied to the results of the learning-based matcher.**

## 12 IMPROVING ACCURACY USING RULES

Though we received positive feedback from the UMETRICS team, we had an internal discussion on how to improve the precision of our matcher. We decided to apply hand-crafted rules to the output of our learning-based matcher. The rules would allow us to make "localized changes". This hopefully would improve precision without reducing recall a lot.

Specifically, we wanted to solicit domain-specific rules from the domain experts (the UMETRICS team) to reduce the number of false positives, then apply these rules to the predictions of our learning-based matcher. To do so, we had an email conversation with the UMETRICS team to understand how to reduce the number of false positives. The UMETRICS team examined the predicted matches, then defined a negative rule (i.e., the rule will flip matches to non-matches), which we discuss next.

**The Negative Matching Rule:** This rule states that a record pair is considered a non-match if one of the following conditions is satisfied:

- Award numbers from UMETRICS and USDA are "comparable," (defined below), and they are not the same.
- Award number from UMETRICS and project number from USDA are "comparable," and they are not the same.

[10]The set of 400 labeled pairs consists of 92 "Yes", 292 "No", and 16 "Unsure". The estimation procedure ignores the "Unsure" pairs.

Here “comparable” means that the award numbers are considered (for this rule) only if they have the same pattern. For example, the UMETRICS award number “03-CS-112313000-031” and the USDA award number “2001-34101-10526” are not comparable because they follow different patterns (i.e., “##-XX-#########-###,” and “YYYY-#####-#####”, respectively, where “#” is any number, “X” is any character, and “YYYY” is a four-digit year).

The UMETRICS award number “WIS01560” and the USDA project number “WIS04509” are comparable because they follow the same pattern “WIS#####” (but because the values are different this pair will be considered a non-match). The UMETRICS team gave us the list of possible patterns for the award numbers from UMETRICS and USDA as well as the project numbers from USDA (not shown for space reasons).

**Applying the Negative Matching Rule:** We applied the negative rule (provided by the UMETRICS team) to the matches and estimated the accuracy again. Conceptually, the learning-based matcher followed by rules could be considered just another matcher like the IRIS matcher. The updated EM procedure is shown in Figure 10. Here, we applied the negative rules to the sets of matches $R_1$ and $R_2$ (obtained from the learning-based matcher). The final set of matches is the union of $C_1$, $D_1$, $S_1$, and $S_2$.

We consider this new workflow a new matcher. Because the candidate set of this new matcher is the same as that of the learning-based matcher from our previous iteration, we can reuse the labeled set (of 400 pairs). Again, we used the Corleone approach to estimate the new precision and recall.

We found that our new matcher (decision tree followed by negative rules) had precision (96.7%, 98.8%) and recall (94.2%, 97.05%). In contrast, the learning-based matcher (without negative rules) had precision (75.2%, 80.3%) and recall (98.1%, 99.6%). The IRIS matcher had the precision (100%, 100%) and recall (65.1%, 71.8%). Thus, compared to the IRIS matcher, our learning-based matcher followed by rules has slightly lower precision, but much higher recall. The final result set has 845 matches, and was shared with the UMETRICS team in a CSV file that uses “UniqueAwardNumber”and “Accession Number” pairs to capture the matches.

The UMETRICS team was delighted with the result, and their director sent the following email:

> *That is really stupendous news! I’m surprised to see how much you were able to raise the precision and recall ... Thanks for all your brilliant work on this.*

**The Next Steps:** As the next immediate step, the UMETRICS team wanted us to package the matcher so that they could move it into the UMETRICS repository to do matching for other data slices. It is similar to moving a workflow that was developed in the development stage into production.

This step raises three challenges. First, the EM workflow is rather complex. It has rules at multiple places (to find sure matches and to update the predictions from the learning-based matcher) and a machine learning-based matcher. So we need to find out how to represent it effectively. Second, the new data may be dirty, so we need to monitor the accuracy of the match results[11]. Finally, if the accuracy is not good enough, we need a way to move back to the development stage and update the EM workflow. Currently, we are working with the UMETRICS team to address these challenges.

## 13 CHALLENGES FOR CURRENT ENTITY MATCHING SOLUTIONS

The end-to-end EM case study that we have just described raises many challenges for the current EM solutions and systems. In what follows we discuss the main challenges[12].

**The Need for How-To Guides:** It should be clear from the case study that it is extremely hard, if not impossible, to fully automate the EM process, end to end. The fundamental reason is because at the start, the user does not even fully understand the data, the match defition, and even what he or she wants.

Here, for instance, initially the users were not aware that some records were missing, or that the match definition was incomplete, or that they actually wanted to match at cluster level, and more. *As a result, most EM projects are really a “conversation” between the EM team and the domain expert team, and this conversation moves forward as new results were produced and discussed.*

If this is the case, then it follows that it is critical to have some how-to guides that tell both teams how to conduct this conversation, what to do first, what to do second, and so on.

Such guides are completely missing from most current EM solutions and systems. While PyMatcher does have an initial how-to guide, as this case study makes clear, that guide is still quite preliminary. It does not provide guidance to many steps such as how to converge on a match definition, and how to collaboratively label effectively, among others. In practice, guides are likely to be complex, as users want to do so many different things, and complications will arise (as seen in this study).

**Many New Pain Points:** Current EM work has largely focused on blocking and matching. This study makes clear that there are many pain points, i.e., *steps that require a lot of work from users*, that current EM research has ignored or not been aware of. For example, how to effectively explore, understand, and clean the tables? While data exploration and cleaning have received significant attention, most of this work has been carried out “in isolation”, independently of work in EM, based on the implicit argument that these problems are orthogonal to EM. We believe, however, that *these problems should be solved in an EM-centric way*, i.e., we should not just try to understand the tables for understanding’s sake, but rather to understand *only things that are important for subsequent EM.*

Other examples of pain points, which require a lot of work from users, include how to quickly converge to a match definition, how to label collaboratively, and how to update an EM workflow if something (e.g., data, match definition) has changed. We argue that more effort should be devoted to addressing these real pain points in practice.

**Different Solutions for Different Parts of the Data:** The vast majority of current EM works treat the input data as of uniform quality, but in practice, this is rarely the case. Instead, the data commonly contains dirty data of varying degree, incorrect

[11]This is typically done by taking a random sample of the predicted matches at regular intervals, manually labeling it, then using the labeled sample to estimate the accuracy. See [28] for an example of monitoring the accuracy of an e-commerce product classification system in production.

[12]In addition to this case study, in the past 3.5 years we have also worked on more than 20 other EM cases for 12 companies and domain science groups [14]. The challenges that we discuss here are not specific to the case stufy of this paper. They arise in many of these other EM cases as well. Further, even though we used PyMatcher in this case study, given the detailed examination of 33 other free and paid EM tools conducted in [18], we believe many of the challenges discussed here would also arise if we were to use these other EM tools.

data, and incomplete data that even domain experts cannot match. It makes no sense trying to debug the system, then spending more time and money to match incorrect and incomplete data. As a result, it is important to have tools that help the user explore and understand the data, then ways to help the user "split" the data into different parts and develop different EM strategies for different parts of the data, as illustrated in this case study (e.g., see Figure 10).

**Support for Easy Collaboration:** We found that in many EM settings there is actually a team of people wanting to work on the problem. Most often they collaborate to label a data set, debug, clean the data, etc. For example, in this case study the UMETRICS team collaboratively labeled and helped debug the labeled data. However, most current EM tools are rudimentary in helping users collaborate easily and effectively. Since users often sit in different locations, it is important that such tools are cloud-based, to enable easy collaboration.

**Handling Changes Along the Way:** No matter how careful we are at the start, changes will likely arise along the way, as the users gain more knowledge and may change their mind. So any effective EM solution need to have good ways to handle such changes. This case study suggested a way to do so, by minimally modifying existing EM workflows and patching them by adding more EM workflows. More research is necessary to evaluate and develop solutions for this problem.

**Managing Machine Learning "in the Wild":** It is clear that ML can be quite effective. This case study suggests that it can help significantly improve recall while retaining high precision, compared to rule-based EM solutions. But the study also shows that deploying even the simplest ML method "in the wild" raises all kinds of challenges, such as labeling, coping with new data, etc. Further, the study also suggests that *the best EM solutions are likely to involve a combination of ML and rules (such as the negative matching rule in this study).*

**Designing EM System Architectures:** Finally, this case study raises fundamental questions about what should be the "right" EM system architecture. It implies that a stand-alone monolithic EM system is not likely to work well, because the users often want to try so many different things and many complications often arise.

Observe that in this study, every time there was a complication, we had to devise a new EM workflow and then wrote Python script (that uses PyMatcher's commands) to implement the new workflow. This is more difficult to do with a stand-alone systems. Instead, the study suggests that an "open-world" EM architecture, such as the one PyMatcher has adopted, where the system is a set of tools that interoperate with one another and also with data science tool in PyData, is more promising. But far more studies and research are necessary to settle this question. For a more detailed discussion on this topic, see our recent work [14], which discusses the system aspects of Magellan.

## 14 CONCLUSIONS

In this paper we have described in detail a case study of entity matching, from raw data to the matches. We have highlighted aspects that previous EM work has ignored. Our case study clearly demonstrates many challenges for current EM work and systems. We hope that this case study on how "sausage is made" can help EM researchers understand better the challenges of the EM process, and thus develop more effective EM solutions.

**Acknowledgments:** We thank the anonymous reviewers for invaluable suggestions that significantly improve this paper. This work is supported by UW-Madison UW2020 grant and NSF grant IIS-1564282, and by Google, Informatica, Johnson Controls, American Family Insurance, and WalmartLabs.

## REFERENCES

[1] [n. d.]. pandas-profiling. https://github.com/pandas-profiling/pandas-profiling.
[2] [n. d.]. Universities: Measuring the Impacts of Research on Innovation, Competitiveness, and Science. https://www.btaa.org/research/umetrics.
[3] Foto N Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, and Jeffrey D Ullman. 2012. Fuzzy joins using MapReduce *(ICDE)*.
[4] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. 2006. Adaptive blocking: Learning to scale up record linkage *(ICDM)*.
[5] Luiz Fernando Carvalho, Alberto Laender, and Wagner Meira Jr. 2015. Entity Matching: A Case Study in the Medical Domain. *CEUR Workshop Proceedings* 1378 (05 2015).
[6] Peter Christen. 2012. *Data Matching.* Springer.
[7] Peter Christen. 2012. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE TKDE* 24, 9 (2012), 1537–1555. https://doi.org/10.1109/TKDE.2011.127
[8] Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. 2016. Distributed Data Deduplication. *PVLDB* 9, 11 (2016), 864–875.
[9] Anish Das Sarma, Ankur Jain, Ashwin Machanavajjhala, and Philip Bohannon. 2012. An Automatic Blocking Mechanism for Large-scale De-duplication Tasks *(CIKM)*. 10. https://doi.org/10.1145/2396761.2398403
[10] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2017. DeepER–Deep Entity Resolution. *arXiv preprint arXiv:1710.00597* (2017).
[11] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate Record Detection: A Survey. *IEEE TKDE* 19, 1 (2007), 1–16.
[12] Mark Flood, John Grant, Haiping Luo, Louiqa Raschid, Ian Soboroff, and Kyungjin Yoo. 2016. Financial entity identification and information integration (feiii) challenge: the report of the organizing committee. In *Proceedings of the Second International Workshop on Data Science for Macro-Modeling*. ACM, 1.
[13] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching *(SIGMOD)*.
[14] Yash Govind et al. 2019. Entity Matching Meets Data Science: A Progress Report from the Magellan Project. UW-Madison Technical Report.
[15] Anitha Kannan, Inmar E. Givoni, Rakesh Agrawal, and Ariel Fuxman. 2011. Matching Unstructured Product Offers to Structured Product Specifications. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.
[16] Lars Kolb, Andreas Thor, and Erhard Rahm. 2012. Dedoop: efficient deduplication with Hadoop. *PVLDB* 5, 12 (2012), 1878–1881.
[17] Pradap Konda et al. 2016. Magellan: Toward Building Entity Matching Management Systems. *PVLDB* 9, 12 (2016), 1197–1208.
[18] Pradap Konda et al. 2016. Magellan: Toward Building Entity Matching Management Systems. Technical Report, http://www.cs.wisc.edu/~anhai/papers/magellan-tr.pdf.
[19] Pradap Konda et al. 2019. Executing Entity Matching End to End: A Case Study. Technical report pages.cs.wisc.edu/~anhai/papers/umetrics-tr.pdf.
[20] Pradap Konda et al. 2019. The UMETRICS Entity Matching Problem: Data, Documentation, and Matches. Available from Magellan's homepage: sites.google.com/site/anhaidgroup/projects/magellan.
[21] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of Entity Resolution Approaches on Real-world Match Problems. *Proc. VLDB Endow.* (Sept. 2010).
[22] Hanna Köpcke, Andreas Thor, Stefan Thomas, and Erhard Rahm. 2012. Tailoring Entity Resolution for Matching Product Offers. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM.
[23] Han Li et al. 2018. MatchCatcher: A Debugger for Blocking in Entity Matching. In *EDBT*.
[24] Xiaochun Li and Changyu Shen. 2013. Linkage of patient records from disparate sources. *Statistical Methods in Medical Research* 22, 1 (2013), 31–38.
[25] Wes McKinney. 2011. pandas: a Foundational Python Library for Data Analysis and Statistics. In *PyHPC*.
[26] Lee Peters, Joan E Kapusnik-Uner, Thang Nguyen, and Olivier Bodenreider. 2011. An approximate matching method for clinical drug names. In *AMIA Annual Symposium Proceedings*, Vol. 2011. American Medical Informatics Association, 1117.
[27] Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB J.* 10, 4 (2001), 334–350.
[28] Chong Sun et al. 2014. Chimera: Large-Scale Classification using Machine Learning, Rules, and Crowdsourcing. *PVLDB* 7, 13 (2014), 1529–1540.
[29] Rares Vernica, Michael J. Carey, and Chen Li. 2010. Efficient Parallel Set-similarity Joins Using MapReduce *(SIGMOD)*. 12. https://doi.org/10.1145/1807167.1807222
[30] B. Weinberg, J Owen-Smith, R. Rosen, L. Schwarz, B. Allen, R. Weiss, and J. Lane. 2014. Science Funding and Short-Term Economic Activity. *Science* 344, 6179 (2014), 41–43.