Magellan: Toward Building Entity Matching Management Systems

Presented by: Pradap Konda
February 27, 2018
**Entity Matching**

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave Smith</td>
<td>Madison</td>
<td>WI</td>
</tr>
<tr>
<td>Joe Wilson</td>
<td>San Jose</td>
<td>CA</td>
</tr>
<tr>
<td>Dan Smith</td>
<td>Middleton</td>
<td>WI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>David D. Smith</td>
<td>Madison</td>
<td>WI</td>
</tr>
<tr>
<td>Daniel W. Smith</td>
<td>Middleton</td>
<td>WI</td>
</tr>
</tbody>
</table>

- Lot of work in this area over the past few decades
- Mainly focus on developing algorithms
Need More Effort on Building EM Systems

- Truly critical to advance the field
- EM is engineering by nature
- Can’t keep developing EM algorithms in vacuum
  - Akin to continuing to develop join algorithms without rest of RDBMS
- Must build systems to evaluate algorithms, integrate R&D efforts, make practical impacts
- As examples, RDBMSs and Big Data systems were critical to advancing their respective fields

But what kind of systems we should build, and how?
Current Research / System Building Agenda for Entity Matching

- Two fundamental steps: blocking and matching

**Table A**

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>Dave Smith</td>
<td>WI</td>
</tr>
<tr>
<td>a₂</td>
<td>Joe Wilson</td>
<td>CA</td>
</tr>
<tr>
<td>a₃</td>
<td>Dan Smith</td>
<td>WI</td>
</tr>
</tbody>
</table>

**Table B**

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>b₁</td>
<td>David D. Smith</td>
<td>WI</td>
</tr>
<tr>
<td>b₂</td>
<td>Daniel W. Smith</td>
<td>WI</td>
</tr>
</tbody>
</table>

Block on state = state $\rightarrow (a₁, b₁) \rightarrow (a₁, b₂) \rightarrow (a₃, b₁) \rightarrow (a₃, b₂) \rightarrow$ match $\rightarrow (a₁, b₁) + (a₁, b₂) - (a₃, b₁) - (a₃, b₂) +$
Current Research / System Building Agenda for Entity Matching

Focus on these two steps
• Develop algorithms
• Maximize accuracy, minimize cost
Assume other steps are trivial

Build stand-alone monolithic systems (e.g., in Java)

Blocker 1  Matcher 1
Blocker 2  Matcher 2
...  ...
This is Far from Enough for Handling EM in Practice

- **EM in practice is significantly more complex**
  - A messy, iterative, multiple-step process
  - Many steps perceived trivial are actually quite difficult to do

- **Even if we let a human user be in charge of the whole EM process, he/she often doesn’t know what to do**

- **Will illustrate in the next few slides**
  - Using an example of applying supervised learning to do EM
How Is EM Done Today in Practice?

- **Development stage**
  - finds an accurate workflow, using data samples

- **Production stage**
  - executes workflow on entirety of data
  - focuses on scalability

1M tuples → block → match (using supervised learning)
Development Stage

Select the best blocker: X, Y

Select the best matcher: U, V
Production Stage

Scaling, quality monitoring, exception handling, crash recovery, …
Limitations of Current EM Systems

- Examined 33 systems
  - 18 non-commercial and 15 commercial ones

1. Do not cover the entire EM workflow
2. Hard to exploit a wide range of techniques
   - Visualization, learning, crowdsourcing, etc.
3. Do not distinguish development vs production stages
4. Very little guidance for users
5. Not designed from scratch for extendability
## Characteristics of 18 Non-Commercial Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
<th>Scenarios</th>
<th>Blocking</th>
<th>Matching</th>
<th>Exploration, cleaning</th>
<th>User interface</th>
<th>Language</th>
<th>Open source</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Atlas</td>
<td>University of Southern California</td>
<td>Single table, two tables</td>
<td>Hash-based</td>
<td>ML-based (decision tree)</td>
<td>No</td>
<td>GUI, commandline</td>
<td>Java</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BigMatch</td>
<td>US Census Bureau</td>
<td>Single table, two tables</td>
<td>Attribute equivalence, rule-based</td>
<td>Not supported</td>
<td>No</td>
<td>Commandline</td>
<td>C</td>
<td>No</td>
<td>Yes (supports parallelism on a single node)</td>
</tr>
<tr>
<td>D-Dupe</td>
<td>University of Maryland</td>
<td>Single table, two tables</td>
<td>Attribute equivalence</td>
<td>Relational clustering</td>
<td>GUI</td>
<td>C#</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dedoop</td>
<td>University of Leipzig</td>
<td>Single table</td>
<td>Attribute equivalence, sorted neighborhood</td>
<td>ML-based (decision tree, logistic regression, SVM etc.)</td>
<td>No</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td>Yes (Hadoop)</td>
</tr>
<tr>
<td>Dedupe</td>
<td>DataMade</td>
<td>Single table, two tables</td>
<td>Canopy clustering, predicate-based</td>
<td>Agglomerative hierarchical clustering-based</td>
<td>Yes</td>
<td>Commandline</td>
<td>Python</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DuDe</td>
<td>University of Potsdam</td>
<td>Single table, two tables</td>
<td>Sorted neighborhood</td>
<td>Rule-based</td>
<td>Yes</td>
<td>Commandline</td>
<td>Java</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Febrl</td>
<td>Australian National University</td>
<td>Single table, two tables</td>
<td>Fall index, blocking index, sorting index, suffixarray index, qgram index, canopy index, stringmap index</td>
<td>Fellegi-Sunter, optimal threshold, k-means, FarthestFirst, SVM, TwoStep</td>
<td>Yes</td>
<td>GUI, commandline</td>
<td>Python</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FRIL</td>
<td>Emory University</td>
<td>Single table, two tables</td>
<td>Attribute equivalence, sorted neighborhood</td>
<td>Expectation maximization</td>
<td>Yes</td>
<td>GUI</td>
<td>Java</td>
<td>Yes</td>
<td>Yes (supports parallelism on a single node)</td>
</tr>
<tr>
<td>MARLIN</td>
<td>University of Texas at Austin</td>
<td>Single table, two tables</td>
<td>Canopy clustering</td>
<td>ML-based (decision tree, SVM)</td>
<td>No</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Merge Toolbox</td>
<td>University of Duisburg-Essen</td>
<td>Single table, two tables</td>
<td>Attribute equivalence, canopy clustering</td>
<td>Probabilistic, expectation maximization</td>
<td>No</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NADEEF</td>
<td>Qatar Computing Research Institute</td>
<td>Single table, two tables</td>
<td>Rule-based</td>
<td>No</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OYSTER</td>
<td>University of Arkansas</td>
<td>Single table, two tables</td>
<td>Attribute equivalence</td>
<td>Rule-based</td>
<td>Yes</td>
<td>Commandline</td>
<td>Java</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>pydedupe</td>
<td>GPoulter (GitHub username)</td>
<td>Single table, two tables</td>
<td>Attribute equivalence</td>
<td>ML-based, rule-based</td>
<td>Yes</td>
<td>Commandline</td>
<td>Python</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RecordLinkage</td>
<td>Institute of Medical Biostatistics, Germany</td>
<td>Single table, two tables</td>
<td>Attribute equivalence</td>
<td>ML-based, probabilistic</td>
<td>Yes</td>
<td>Commandline</td>
<td>R</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SERF</td>
<td>Stanford University</td>
<td>Single table</td>
<td></td>
<td>R-Swoosh algorithm</td>
<td>No</td>
<td>Commandline</td>
<td>Java</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Silk</td>
<td>Free University of Berlin</td>
<td>RDF data</td>
<td>Rule-based</td>
<td>Yes</td>
<td>GUI</td>
<td>Java</td>
<td>Yes</td>
<td>No</td>
<td>Yes (supports parallelism on a single node, Hadoop)</td>
</tr>
<tr>
<td>TAILOR</td>
<td>Purdue University</td>
<td>Single table, two tables</td>
<td>Attribute equivalence, sorted neighborhood</td>
<td>Probabilistic, clustering, hybrid, induction</td>
<td>No</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WHIRL</td>
<td>William Cohen</td>
<td></td>
<td></td>
<td>Vector space model</td>
<td>Commandline</td>
<td>C++</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
## Characteristics of 15 Commercial Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose and how EM fits in</th>
<th>Supported EM scenarios</th>
<th>Main user interface</th>
<th>Distinction between dev. and prod. stages</th>
<th>Language</th>
<th>Scripting environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataMatch from Data Ladder</td>
<td>Data cleaning, data matching. EM forms the core of their solution</td>
<td>Multiple tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Dedupe.io</td>
<td>Record linkage, deduplication. EM forms the core of their solution</td>
<td>Single table, two tables</td>
<td>Web-based</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>FuzzyDups</td>
<td>Duplicate detection, data cleaning. EM forms the core of their solution</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>GraphLab Create</td>
<td>EM is offered as a service on top of their GraphLab platform</td>
<td>Single table, two tables, linking records to a KB</td>
<td>Web-based</td>
<td>C++</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>IBM InfoSphere</td>
<td>Customer data analytics. EM is supported by a component (BigMatch) in the product</td>
<td>Single table, two tables</td>
<td>Web-based</td>
<td>Java</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Informatica Data Quality</td>
<td>Improve data quality. EM forms a part of data quality pipeline</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LinkageWiz</td>
<td>Data matching and data cleaning. EM forms the core of their solution</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oracle Enterprise Data Quality</td>
<td>Improve data quality. EM forms a part of data quality pipeline</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentaho Data Integration</td>
<td>ETL, data integration. EM forms a part of ETL/data integration pipeline</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>Java</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SAP Data Services</td>
<td>Improve data quality, data integration. EM forms a part of data integration pipeline</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS Data Quality</td>
<td>Improve data quality. EM forms a part of data quality pipeline</td>
<td>Single table, multiple tables</td>
<td>Web-based</td>
<td>Limited support</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strategic Matching</td>
<td>Data matching and data cleaning. EM forms the core of their solution</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Talend Data Quality</td>
<td>Improve data quality. EM forms a part of data quality pipeline</td>
<td>Single table, two tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tamr</td>
<td>Data curation. EM forms a part of data curation pipeline</td>
<td>Multiple tables</td>
<td>Web-based</td>
<td>Java</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Trillium Data Quality</td>
<td>Improve data quality. EM forms a part of data quality pipeline</td>
<td>Single table, multiple tables</td>
<td>GUI</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Do Not Cover the Entire EM Workflow

- Focus on blocking and matching
  - Develop ever more complex algorithms
  - Maximize accuracy and minimize costs

- Assume other steps are trivial
  - In practice these steps raise serious challenges
  - Example 1: sampling to obtain two smaller tables A’ and B’
  - Example 2: sample a set of tuple pairs to label
  - Example 3: label the set
Select the best blocker

Select the best matcher

A

B

A'

B'

A'

B'

A'

B'

A'

B'

Development Stage

A sample → A' sample → A' blocker → X blocker → C_x

A sample → A' sample → A' blocker → Y blocker → C_y

matcher → (-,-) matcher → (-,-) + matcher → (-,-) - matcher → (-,-) +

quality check no yes

S

G

cross-validate matcher U

0.89 F_1
cross-validate matcher V

0.93 F_1

Select the best blocker

Select the best matcher
Example 1: Sampling Two Smaller Tables

● Tables A and B each has 1M tuples
  – Very difficult to experiment with them directly in development stage
  – Way too big, so too time consuming

● Need to sample smaller tables
  – A’ from A, B’ from B, say 100K tuples for each table

● How to sample?
  – Random sampling from A and B may result in very few matching tuple pairs across A’ and B’
  – How to resolve this?
Example 2: Take a Sample from the Candidate Set (for Subsequent Labeling)

- Let C be the set of candidate tuple pairs produced by applying a blocker to two tables A’ and B’

- We need to take a sample S from C, label S, then use the labeled set to find the best matcher and train it

- How to take a sample S from C?
  - Random sampling often does not work well if C contains few matches
  - In such cases S contains no or very few matches
Example 3: Labeling the Sample

- This task is often divided between two or more people.

- As they label their set of tuple pairs, they may follow very different notions of matching:
  - E.g., given two restaurants with same names, different locations
  - A person may say “match”, another person may say “not a match”

- At the end, it becomes very difficult to reconcile different matching notions and relabel the sample.

- This problem becomes even worse when we crowdsource the labeling process.
An Illustrating Example for Distributed Labeling

Two restaurants match if they refer to the same real-world restaurant

([Laura’s, 23 Farewell Str], [Laura, 23 Farewell]) +
([Palmyra, 46 Main St], [Palmyra, 15 Broadway]) -
([KFC, 24 Main St], [KFC, 41 Johnson Ave]) +
2. Hard to Exploit a Wide Range of Techniques

- EM steps often exploit many techniques
  - SQL querying, keyword search, learning, visualization, information extraction, outlier detection, crowdsourcing, etc.

- Difficult to incorporate all into a single system

- Difficult to move data repeatedly across systems
  - An EM system, a visualization system, an extraction system, etc.

- Problem: most systems are stand-alone monoliths, not designed to play well with other systems
3. Do Not Distinguish Dev vs Prod Stages

- Current systems
  - Provide a set of blockers / matchers
  - Provide a way to specify / optimize / execute workflows
  - Pay very little attention to the development stage
4. Little Guidance for Users

- Suppose user wants at least 95% precision & 80% recall

- How to start? With rule-based EM first? Learning-based EM first?

- What step to take next?

- How to do a step?
  - E.g., how to label a sample?

- What to do if after much effort, still hasn’t reached desired accuracy?
5. Not Designed from Scratch for Extendability

- Can we build a single system that solves all EM problems?
  - No

- In practice, users often want to
  - Customize, e.g., to a particular domain
  - Extend, e.g., with latest technical advances
  - Patch, e.g., writing code to implement lacking functionalities

- Users also want interactive scripting environments
  - For rapid prototyping, experimentation, iteration

- Most current EM systems
  - Are not designed so that users can easily customize, extend, patch
  - Are not situated in interactive scripting environments
Many serious limitations:

1. Do not cover the entire EM workflow
2. Hard to exploit a wide range of techniques
3. Do not distinguish development vs production stages
4. Very little guidance for users
5. Not designed from scratch for extendability
Our Solution: The Magellan Approach

- **Define a clear scope**
  - Each system targets a set of EM scenarios and power users

- **Solve the development stage**
  - Develop a how-to guide
    - Helps users discover accurate workflow
    - Must cover all steps
    - Tells users what to do, step by step
  - Develop tools for pain points in the guide
    - On top of PyData ecosystem

- **Solve the production stage in a similar way**
  - But focus on scalability, crash recovery, etc.
How-to Guide/Tools for Development Stage

Select the best blocker

Select the best matcher
1. Load tables A and B into Magellan. Downsample if necessary.

2. Perform blocking on the tables to obtain a set of candidate tuple pairs C.

3. Take a random sample S from C and label pairs in S as matched / non-matched.

4. Create a set of features then convert S into a set of feature vectors H. Split H into a development set I and an evaluation set J.

5. Repeat until out of debugging ideas or out of time:
   
   (a) Perform cross validation on I to select the best matcher. Let this matcher be X.

   (b) Debug X using I. This may change the matcher X, the data, labels, and the set of features, thus changing I and J.

6. Let Y be the best matcher obtained in Step 5. Train Y on I, then apply to J and report the matching accuracy on J.
How-to Guide/Tools for Development Stage

Users want step-by-step guide on how to take a sample then label it

Tool to highlight possible matching categories

Tool to debug labels

Tool to help revise labels

([Laura’s, 23 Farewell Str], [Laura, 23 Farewell]) +
([Palmyra, 46 Main St], [Palmyra, 15 Broadway]) -
([KFC, 24 Main St], [KFC, 41 Johnson Ave]) +
Build Tools on the PyData Ecosystem

- **Key observation**
  - Development stage does a lot of data analysis
    - E.g., analyzing data to discover EM matching rules
    - Often requires cleaning, visualizing, finding outliers, etc.
  - Very hard to incorporate all such techniques into a single EM system
  - Better to build on top of an open-source data ecosystem

- **Two major current ecosystems**
  - Python and R

- **PyData ecosystem**
  - Used extensively by data scientists
  - > 100K packages (in PyPI)
  - Data analysis stack / big data stack
The Magellan Architecture

Development Stage

How-to guide

Tools for pain points
(as Python commands)

Data samples

Match two tables

Production Stage

How-to guide

Tools for pain points
(as Python commands)

Original data

Power Users

EM Workflow

Data Analysis Stack

pandas, scikit-learn, matplotlib, numpy, scipy, pyqt, seaborn, …

Big Data Stack

PySpark, mrjob, Pydoop, pp, dispy, …

Python Interactive Environment
Script Language

PyData ecosystem
Raises Numerous R&D Challenges

- Developing good how-to guides is very difficult
  - Even for very simple EM scenarios

- Developing tools to support how-to guides raises many research challenges
  - Accuracy, scaling

- Novel challenges for designing open-world systems
Examples of Current Research Problems

- Profile the two tables to be matched, to understand different matching definitions
- Normalize attribute values using machine and humans
- Verify attribute values using crowdsourcing
- Debug the blocking / labeling / matching process
- Scale up blocking to 100s of millions of tuples
- Apply Magellan template to string similarity joins
- ...

- Our group is working on many of the above challenges
Magellan has been built as an open-world system
  – On top of Python data ecosystem
  – Relies on external systems to supply tools in visualization, mining, IE, etc.

Raises many non-trivial challenges
  – Managing metadata
  – Designing data structures
  – Handling missing values
  – Package version incompatibilities
  – Data type mismatch
  – …
Metadata Management

Closed-World Systems

RDBMS

SQL queries
commands

data

A
B

metadata

A.ssn is a key

Open-World Systems

Magellan

command 1
command 2
...
data

A
B

metadata

A.ssn is a key

System X

command x₁
command x₂
...

System Y

command y₁
command y₂
...

data

C

metadata

...

PyData ecosystem
Metadata Management: Naïve Solution

- Rewrite the external commands to be metadata aware

- **Issues:**
  - Need a lot of developer effort
    - Impractical given the large number of commands and packages that users can use
  - Cannot force the user to wait till a developer has made an external command metadata aware
Metadata Management: Current Solution

- Design each command in Magellan to be metadata aware
- Each command at the start, checks for all the metadata constraints that it requires to be true
  - E.g. primary key constraint must be satisfied to operate on Table A
  - Command will not proceed until all the required constraints are satisfied
- During its execution it will try to manage metadata properly
- If it encounters an invalid constraint, it will alert the user
  - But will continue execution as the constraint is not critical for the correct execution
Designing Data Structures

- **At the heart of Magellan is a set of tables**
  - Tuples to be matched are stored in two tables A and B
  - Intermediate and final results can also be stored in tables
  - Need to store metadata
  - Important to study how to implement tables

- **Design alternatives**
  - Use Pandas data frame to store and process tables
  - Define a new class with multiple fields.
    - One field stores the data frame and other fields store metadata
  - Subclass Pandas data frame and add fields to store metadata
Alternative 1: Use Pandas Data Frame

- Pandas is a popular package to store and process tables using data frame data structure

- Naïve solution is to implement Magellan tables as Pandas data frames

- Issue: cannot store metadata
  - E.g. primary key of a table
Alternative 2: Include Pandas Data Frame in Another Class

- Define a new class, MTable and include a field for Pandas data frame and other fields for metadata

- Issues:
  - Makes it difficult for other packages operate on Magellan’s data
    - Existing packages are completely unaware of MTable
    - Commands in these packages cannot operate on MTable objects directly
  - Need to redefine commands from other packages, a time-consuming and brittle process
### Alternative 3: Inherit Pandas Data Frame

- **Subclass Pandas data frame to define a new class MDataframe**
  - Include fields to store metadata
- **Any existing command that knows data frames can operate on MDataframe objects**

- **Issue:**
  - Commands may return inappropriate type of objects
    - Can be quite confusing to users
Current Solution: Pandas Data Frame + Catalog

- **Store Magellan tables as Pandas data frames**
  - Any existing Python package that manipulates data frames will work
  - Maximizes the chances of interoperating with other packages seamlessly

- **Store metadata in a separate object, catalog**
  - Similar to RDBMS
  - Stores metadata for each table in Magellan
  - Magellan commands which require metadata can probe the catalog

- **General principle**
  - Use data structures that are most common to other systems to store its data
  - When not possible, provide procedures to convert between its own type and the ones commonly used by other systems
Current Status of Magellan

- Has been in development since June 2015
  - ~18 months
  - 1 main developer + 2 contributors

- Contains 7 major new tools for how-to guides

- Built on top of 11 different packages from PyData ecosystem
  - E.g., Pandas, Scikit-learn, etc.

- Exposes 104 commands to users

- Codebase includes 87 Python files with ~14K lines of Python code
Current Status of Magellan

- **Package is comprehensively tested**
  - 1136 unit test cases
  - 90 performance test cases

- **Codebase is extensively documented**
  - 5K lines of comments

- **Most advanced and comprehensive open-source EM system available today**

- **Used extensively in education, science and at several companies**
Current Status of Magellan

- Used as a teaching tool for data science classes at UW
  - CS 638: 83 students
  - CS 784: 44 students

- Used in biomedicine domain to match drugs
  - 2 accepted posters
  - Highlighted in CPCP 2017

- Used at companies
  - @WalmartLabs
  - Johnson Controls
  - RECRUIT Institute of Technology
  - Marshfield Clinic

- Resulted in a research paper and a demonstration at VLDB ’16
# Experiments with 44 Students

<table>
<thead>
<tr>
<th>Team</th>
<th>Domain</th>
<th>Table A Size</th>
<th>Table B Size</th>
<th>Cand. set Size</th>
<th>First learning Iteration (A)</th>
<th>Final best learning matcher (B)</th>
<th>Num. of Iterations</th>
<th>Final matcher (C)</th>
<th>Num. of Iterations</th>
<th>Diff. in F1 between (C) and (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vehicles</td>
<td>4786</td>
<td>9003</td>
<td>8009</td>
<td>71.2</td>
<td>71.2</td>
<td>91.43</td>
<td>94.12</td>
<td>92.75</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Movies</td>
<td>7391</td>
<td>6408</td>
<td>78079</td>
<td>99.28</td>
<td>95.13</td>
<td>97.04</td>
<td>98.21</td>
<td>100 99.1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Movies</td>
<td>3000</td>
<td>3000</td>
<td>1000000</td>
<td>98.9</td>
<td>99.44</td>
<td>99.5</td>
<td>98.63</td>
<td>98.63</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Movies</td>
<td>3000</td>
<td>3000</td>
<td>36000</td>
<td>68.2</td>
<td>69.16</td>
<td>68.6</td>
<td>98</td>
<td>100 98.99</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Movies</td>
<td>6225</td>
<td>6392</td>
<td>54028</td>
<td>100</td>
<td>95.23</td>
<td>97.44</td>
<td>100 100</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Restaurants</td>
<td>6960</td>
<td>3897</td>
<td>10630</td>
<td>100</td>
<td>37.5</td>
<td>54.55</td>
<td>100 88.89</td>
<td>94.12</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>Electronic Products</td>
<td>4559</td>
<td>5001</td>
<td>823832</td>
<td>73</td>
<td>51</td>
<td>59</td>
<td>73.3</td>
<td>64.71</td>
<td>68.75</td>
</tr>
<tr>
<td>8</td>
<td>Music</td>
<td>6907</td>
<td>55923</td>
<td>58692</td>
<td>92</td>
<td>79.31</td>
<td>85.19</td>
<td>90.48</td>
<td>82.61</td>
<td>86.36</td>
</tr>
<tr>
<td>9</td>
<td>Restaurants</td>
<td>9947</td>
<td>28787</td>
<td>400000</td>
<td>100</td>
<td>78.5</td>
<td>87.6</td>
<td>94.44</td>
<td>97.14</td>
<td>95.77</td>
</tr>
<tr>
<td>10</td>
<td>Cosmetic</td>
<td>11026</td>
<td>6445</td>
<td>36026</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>96.67</td>
<td>87.88</td>
<td>92.06</td>
</tr>
<tr>
<td>11</td>
<td>EBooks</td>
<td>6482</td>
<td>14110</td>
<td>13652</td>
<td>96.67</td>
<td>96.67</td>
<td>96.67</td>
<td>100</td>
<td>95.65</td>
<td>97.78</td>
</tr>
<tr>
<td>12</td>
<td>Beer</td>
<td>4346</td>
<td>3000</td>
<td>4334961</td>
<td>84.5</td>
<td>59.6</td>
<td>65.7</td>
<td>100</td>
<td>60.87</td>
<td>75.68</td>
</tr>
<tr>
<td>13</td>
<td>Books</td>
<td>3506</td>
<td>3508</td>
<td>2016</td>
<td>93.46</td>
<td>100</td>
<td>96.67</td>
<td>91.6</td>
<td>100</td>
<td>95.65</td>
</tr>
<tr>
<td>14</td>
<td>Books</td>
<td>3967</td>
<td>3701</td>
<td>4029</td>
<td>74.17</td>
<td>82.2</td>
<td>82.5</td>
<td>100</td>
<td>84.85</td>
<td>91.8</td>
</tr>
<tr>
<td>15</td>
<td>Anime</td>
<td>4000</td>
<td>4000</td>
<td>138344</td>
<td>95.9</td>
<td>88.9</td>
<td>92.2</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>16</td>
<td>Books</td>
<td>3021</td>
<td>3098</td>
<td>931</td>
<td>74.2</td>
<td>100</td>
<td>85.2</td>
<td>96.34</td>
<td>84.95</td>
<td>90.29</td>
</tr>
</tbody>
</table>

- **Baseline:** \( P = 56-100\%, \ R = 37-100\%, \ F1 = 56-99\% 

- **Magellan:** \( P = 91-100\%, \ R = 64-100\%, \ F1 = 78-100\% 

  - 20 teams out of 24 achieved recall above 90\%
Experiments with 44 Students

- **Tools for pain points were highly effective**
- **Debugging blockers**
  - 18 out of 24 teams used the debugger, for 5 iterations on average
  - Debugger helps in:
    (a) cleaning data
    (b) finding correct blocker types/attributes
    (c) tuning blocker parameters
    (d) knowing when to stop
- **Debugging matchers**
  - Teams performed 3 debugging iterations on average
  - Actions performed include:
    (a) feature selection
    (b) data cleaning
    (c) parameter tuning
- **Students extensively used visualization, extraction, cleaning, etc. (using PyData packages)**
Magellan “in the Wild”

- **WalmartLabs**
  - Helped improve a system already in production

- **Johnson Controls**
  - Matched hundreds of thousands of suppliers for JCI
  - Precision above 95%, recall above 92% (across many data sets)

- **Marshfield Clinic**
  - Matched 18M pairs of drugs
  - Precision: 99.18% Recall: 95.29%

- **Raised additional interesting challenges**
  - Data can be very dirty, need far more cleaning tools
Novelties in the Current Work

● Conceptual novelties:
  – Radically different from current EM systems
  – Conceptually novel architecture and methodology
    ▪ Distinguish between development & production stages
    ▪ Provide how-to guides
    ▪ Identify pain points and develop supporting tools
    ▪ Implement tools on top of the PyData ecosystem

● Technical novelties:
  – Realizing such conceptual novelties raises many research problems
  – Many of them are pursued by members of our group
  – Provided preliminary solution for some of the problems
    ▪ Metadata management, designing data structures

● Practical impact:
  – Magellan has been released as an open-source tool
  – Used in education, science and companies
For More Details

- Check out Magellan under http://pages.cs.wisc.edu/~anhai/
- GitHub: github.com/anhaidgroup