ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

DATA MINING (CO3029)

Assignment Report

Magellan Tool

Mentor: Lê Hồng Trang
Team: 9
Member: Trần Ngọc Bảo Hân – 2013113
Nguyễn Trọng Đức Huy – 2011283
Trần Hà Tuấn Kiệt – 2011493
Nguyễn Tiến Nam – 2011652
Trương Huy Thái – 2012036
Nguyễn Đức Thuỵ – 2012158

Tp. Hồ Chí Minh, Tháng 4/2023

Nội dung

1 Introduction

Enterprises today generate huge amounts of data in their daily operations. Some of it is produced by the sales, marketing, and customer service arms of the business. Other parts may arise from the company's financial transactions, or perhaps its research, development, and production activities. For example, the retail giant Walmart, which is the world's largest company by revenue. Seamlessly integrating data across a large, enterprise retailer with 20,000 brick-and-mortar store locations, a massive online website, millions of items in inventory, mobile apps, global data, and 3rd party resellers becomes yet another level of complexity. Not only do they need to collect data across every customer, store, warehouse, website, and application, they need real-time data integration in order to function properly at scale. To maximize the value insights from big data sources, it has to do data integration process in real-time to have timely analysis for company strategies operation. This is where the term 'data integration' comes into play. Consequently, problems related to data integration, especially entity matching arise.

1.1 What is entity matching?

In order to gain an understanding of entity matching, we should have a look in the data integration process to identify the step in which entity matching is being implemented.

Integrating data from different sources consists of three tasks. The first task is schema matching, which is concerned with identifying database tables, attributes and conceptual structures (such as ontologies, XML schemas and UML diagrams) from disparate databases that contain data that correspond to the same type of information. The second task is data matching, the task of identifying and matching individual records from disparate databases that refer to the same real-world entities or objects. A special case of data matching is *duplicate detection*, the task of identifying and matching records that refer to the same entities within a single database. The third task, known as data fusion, is the process of merging pairs or groups of records that have been classified as matches (i.e. that are assumed to refer to the same entity) into a clean and consistent record that represents an entity. When applied on one database, this process is called deduplication.

The initial idea of record linkage goes back to Halbert L. Dunn in his 1946 article titled "Record Linkage" [1]. Howard Borden Newcombe then laid the probabilistic foundations of modern record linkage theory in a 1959 article in Science. These were formalized in 1969 by Ivan Fellegi and Alan Sunter, in their pioneering work "A Theory For Record Linkage" [2], where they proved that the probabilistic decision rule they described was optimal when the comparison attributes were conditionally independent. **Entity matching is the task of identifying entities (objects, data instances) referring to the same real-world entity** [3]. It also referred to as duplicate identification, record linkage, entity resolution or reference reconciliation. Let's have an example to imagine one of entity matching problems. Given two tables A and B, the goal of EM is to discover the tuple pairs between two tables that refer to the same real-world entities. So entity matching is a second task of data integration process.

We have an example figure 1 that shows us the difficulty in entity matching. We have 2 tables and how we can identify whether Mr. Adam from table 1 and Smith, Adam from table 2 be the same entity. Nowadays, real-world data is rarely in clean and structured condition. Due to manual insertion, some mistakes can appear such as typos, alternative spellings,... Some data might be missing. Additionally, data in entity matching is often assumed to be structured in records. But in practical, data can be in diverse format and not only in : structured, semi-structured and

First Name	Middle Name	Last Name	Phone	Address 1	Address 2	City	State	Country
Mr. Adam	R.	Smith	1-777-888-9999	112 Bell Street	Near Church Tower	Oakland	California	US
Dr. Raksha	Ranganathan		+91-8383838383	21, Brigade Road	Opposite McD	Bengaluru	Karnataka	India

Name	Age	Address	State	Phone
Smith, Adam	33	112, Bell Street, Near Church Tower, Oakland	CA	+1-777-888-9999
PhD. Raksha Rangnathan	40	21, Brigade Road, Opp. McDonalds, Bangalore	Karnataka, India	918383838383

Hình 1: Entity matching problem with 2 tables

unstructured. If we want to do EM, we need to transfer these data to structured data and it is not easy to do that. It is not always reasonable to assume that various data sources will adhere to the same schema, format, and syntactic conventions. That leads to inconsistency and poor data quality. Entity matching problem arised. As a result, some techniques appeared to resolve its problem. Recently, novel approaches utilizing deep learning methods for natural language processing have emerged. We also have neural networks to be a good technique for entity matching problem.

Entity matching is a crucial and difficult task for data integration [3]. This issue has been a subject of continuous study for many years, resulting in various methods being developed. Entity matching can be performed using a range of techniques, including rule-based and machine learning-based approaches. Rule-based approaches rely on a set of predefined rules to identify matches, while machine learning-based approaches learn from labeled data to identify matches. There are many techniques in the world

1.2 Why is entity matching important?

EM has numerous applications in real life with diverse fields. That makes it becomes more important in real life. Some applications are:

- Data science: Data scientists can clean a customer table by detecting duplicate customers, to construct a 360-degree view of customers by linking multiple customer tables, or to conduct a market analysis by comparing the same product across different websites.
- Healthcare: Clean and matched patient data can also be used to create health indicators. Entity matching solution enables the consolidation of a patient's data from various data sources, matching data from hospitals and private clinics, insurance providers and claims, personal profiles, and from the internet and social platforms to create a unique profile of each patient.
- Marketing: customer data is gotten from multiple systems, such as CRM, ERP, or support, into a single environment. Details on past calls, emails, purchases, chat sessions, and various

other activities are added as well. That is, we have a big dataset about our customers. So how we get to know they are the same person with the typos of name and different email. Companies have to conduct entity matching techniques to identify whether that customer is just one person or two different persons. Companies can apply predictive analytics to this wealth of data. The system could then make a right personalized product recommendation to demanding customers, reducing one customer get two similar email at the same time.

- Supplychain: Supplier matching would also enable companies to understand pricing across divisions, find the best supplier for a given product easily, negotiate prices, onboard new suppliers faster, get rid of duplicate suppliers. For instance, let's say a consumer packaged goods (CPG) company wants to compare two suppliers who offer similar products to determine who sells more at a lower price. The vendors may use different names for the same product. One supplier may refer to it as tissue paper, while the other may call it paper handkerchief, tissue, or sanitary paper. As the product names will not match exactly, entity resolution comes into play to reconcile the products, compare their prices, and determine which vendor is offering a better price.

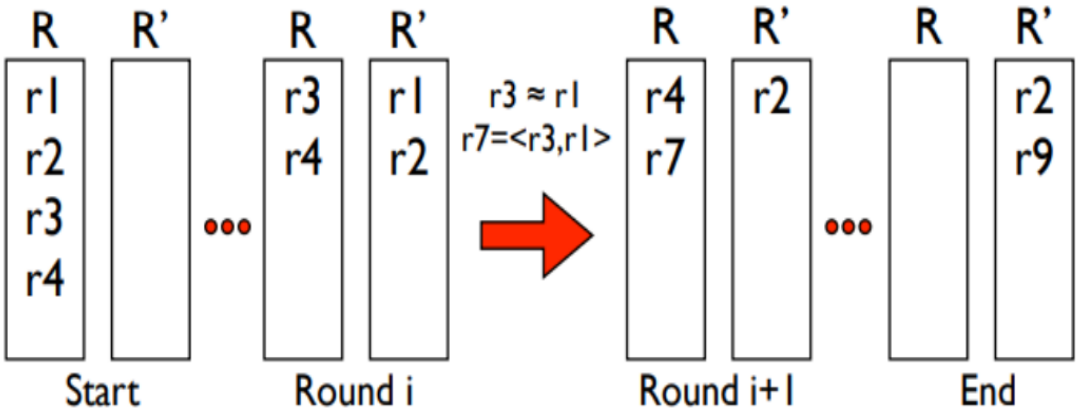
2 Current Solutions

Firstly, we'll discuss the previous and current attempts at trying to solve the entity matching problems and how they lead to our research on Magellan.

2.1 Framework without training

In this section, we'll focus on one of the earliest attempts of solving the entity matching problems, which is **SERF (Stanford Entity Resolution Framework)**, developed by Info Labs at Stanford University in 2004.

1. **Explanations:**
   - SERF is a generic EM infrastructure with the focus on improving the efficiency of entity matching. Due to this reason, there are no detailed sections on the matchers (also known as similarity functions) in most reports written by researchers from Info Labs, but they did mention that the matchers would be considered as "black boxes" waiting to be invoked by the EM engine.
   - The main study that Info Labs did was focusing on different algorithms that could minimize the number of invocations to these potentially expensive "black boxes" (mainly by keeping track of previously compared values).
2. **R-swoosh Algorithm:**
   This is the main algorithm developed in the SERF project and one of the most efficient at the time.
   R-Swoosh relies on two sets: R, which initially contains all the input records, and R0, which maintains the set of (so far) non-dominated, non-matching records. R-Swoosh successively compares each record in R to all the records present in R0. R-Swoosh performs a merge as soon as a pair of matching records is found. The obtained record is added to R, and the pair of matching records is deleted immediately. The algorithm terminates when R is empty, and R0 contains ER(R).



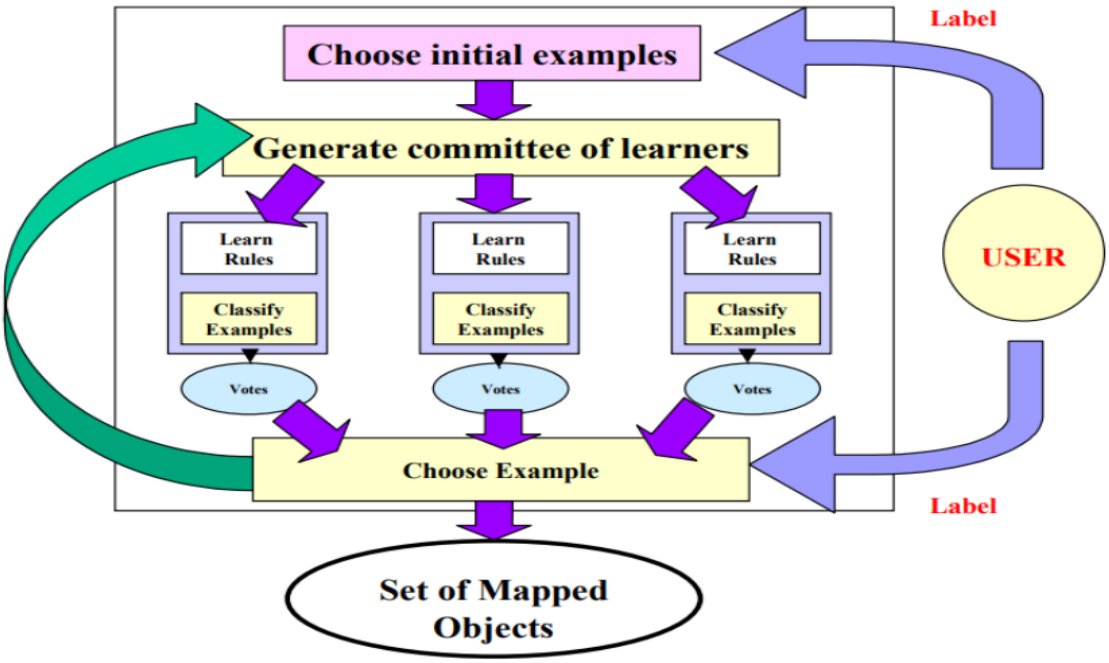
Hình 2: A sample run of the R-swoosh Algorithm

Also, on more advanced researches, the team at Stanford has successfully distributed the system in purpose of reducing the costs which will use a more advanced version of R-swoosh which is called D-swoosh.

2.2 Training-based framework

In this section, we'll focus on the first ever solution of training-based frameworks which is called **Active Atlas** [4], proposed by Prof. Sheila Tejada and her colleagues at the University of Southern California back in 2001

1. **Explanations:**
   - The Active Atlas system allows the training-based determination of matching rules by utilizing a combination of several decision tree learners.
   - The training selection process is semi-automatic to minimize the number of required training examples. This is achieved by letting the decision tree learners vote on the most informative example for users to classify next ("active learning").
   - Attribute value matchers use a variant of TF–IDF (Term Frequency-Inverse Document Frequency) that allows considering a variety of additional information (e.g., stemming, abbreviations) to determine the common tokens of two attribute values. Also, a disjoint blocking strategy based on hashing is supported.
2. **Mapping-rule Learning:** There are 2 types of learning techniques that are implemented in Active Atlas, and we'll go through briefly on them.
   - **Decision Tree Learning:**
     Decision trees are created by determining which are the most useful attributes to classify the examples. A metric called information gain measures how well an attribute divides the given set of training examples by their classification (mapped/not mapped). Creating the decision tree is an iterative process, where the attribute with the greatest information gain is chosen at each level. Once a decision tree is created, it can be converted into mapping rules.
   - **Active Learning:**
     This technique generates a committee of decision tree learners that vote on the most informative example or candidate mapping for the user to classify next. The query by bagging technique is considered an active learning technique because the system actively chooses examples for the user to label. This committee-based approach is used in order to reduce the number of user-labeled examples.



Hình 3: A full mapping-rule learners system

2.3 Hybrid Framework

Hybrid framework is the type of framework which combines the two frameworks above, using supervised matcher combinations as well the manual specification of EM strategies without training. In this section, we'll talk briefly of **FEBRL (Freely Extensible Biomedical Record Linkage)** [5], one of the most used EM system ever since its release in 2005 by researchers at Australian National University (ANU).

1. **Explanations:**
   - FEBRL is mainly a training-based numerical combination approach utilizing the SVM (Support Vector Machine) as well as numerical approaches without training. It was originally developed for entity matching in the biomedical domain (hence the name).
   - There are a large selection of 26 different similarity measures is available for attribute value matching. The system also supports one disjoint as well as three overlapping blocking methods.
   - Besides manual training selection, two strategies for automatic training are supported, which are threshold and nearest-based.
2. **Overall System:** The FEBRL system is implemented in an object-oriented design with a handful of modules, each containing routines for specific tasks.
   (a) *Input Data Initialization:*
   In the first step, a user has to select if she or he wishes to conduct a project for (a) cleaning and standardization of a data set, (b) deduplication of a data set, or (c) linkage of two data sets.
   (b) *Data Exploration:*
   The 'Explore' page allows the user to analyze the selected input data set(s) in order

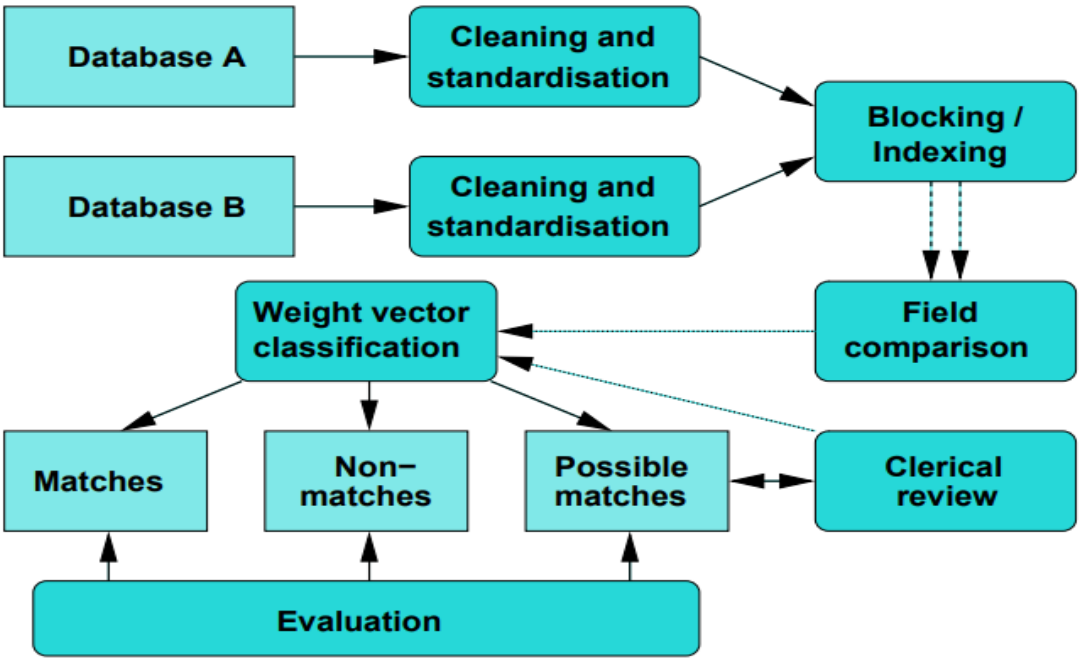
to get a better understanding of the content and quality of the data to be used for a standardization, deduplication or linkage project. In order to speed up exploration of large data sets, it is possible to select a sampling rate as a percentage of the number of records in a data set.

(c) *Data Cleaning and Standardization:*
The cleaning and standardization of a data set using the Febrl GUI is currently done separately from a linkage or deduplication project, rather than as a first step. A data set can be cleaned and standardized and is written into a new data set, which in turn can then be de-duplicated or used for a linkage.

(d) *Indexing (Blocking):*
Once the required input data sets have been initialized, the indexing method and its corresponding index (block) keys have to be defined for a deduplication or linkage. There are multiple indexing methods for users to choose from.

(e) *Weight Vector Classification:*
The last major step required is the selection of the method used for weight vector classification and setting of its parameters. Currently, FEBRL offers a few different classification techniques.

Besides these major modules, there are a few more modules that are also implemented which are "Outputs", "Evaluation & Clerical Review" and "Log Page" but we'll not go into them for now.



Hình 4: General record linkage process

2.4 Pros and Cons of non-commercial systems

1. **Pros:**
   The main advantage of non-commercial systems is that they mainly focus on the Entity Matching step of the whole process. That's why the algorithms, techniques and methods for Entity Matching have grown fast and strong, from generic matching to training with more and more advanced algorithms to combining the two of them together.
2. **Cons:**
   - Mostly focus on the scenarios of matching within a single table or across two tables.
   - No guidance on how to select appropriate blockers and matchers.
   - Some systems provide limited data exploration capabilities (e.g., browsing, showing statistics about the data) and cleaning capabilities (mostly ways to perform relatively simple transformations such as regex-based ones and to clean certain common attributes). No system provides support for less well-known but critical steps such as debugging, sampling, and labeling.
   - No system provides how-to guides that tell users how to do EM, step by step. And no system makes a distinction between the development stage and the production stage.
   - Less than half of the systems are open-source, and no system provides any easy interfacing with data science stacks.
   - About half of the systems provide just command line interfaces, while the remaining half also provide GUIs. A few systems are still providing limited scaling capabilities.

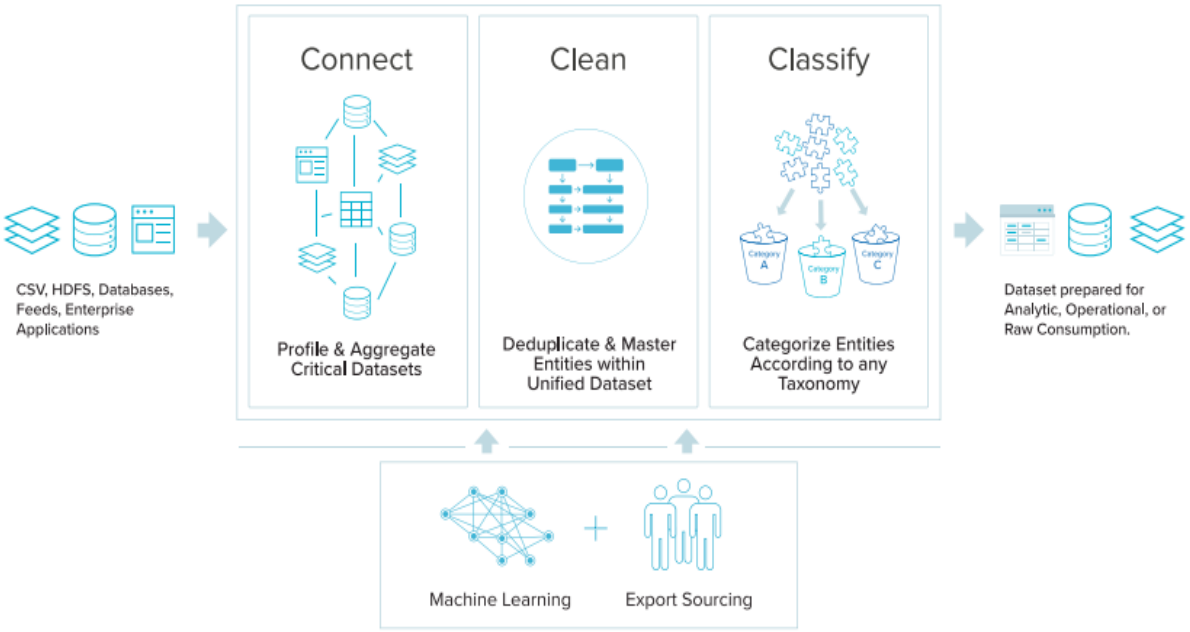
2.5 Commercial Tools

Those that were introduced above are all non-commercial tools, which means that they're not in production for global users. But there are a few commercial ones that are created and utilized for business use cases. In this section, we'll introduce **TAMR**, probably the most used commercial EM system at the moment.

1. **Explanations:**
   This system has entity matching as a component in a data curation pipeline. This EM component effectively does deduplication and merging: given a set of tuples D, clusters them into groups of matching tuples, and then merges each group into a super tuple.
2. **How it works:**
   - Toward the above goal, TAMR starts by performing blocking on the set of tuples D. Specifically, it creates a set of categories, then use some relatively inexpensive similarity measure to assign each tuple in D to one or several categories. Only tuples within each category will be matched against one another.
   - Next, TAMR obtains a set of tuple pairs and asks users to manually label them as matched / non-matched. TAMR takes care to ensure that there are a sufficient number of matched tuple pairs in this set.
   - Next, TAMR uses the labeled data to learn a set of matching rules. These rules use the similarity scores among the attributes of a tuple pair, or the probability distributions of

attribute similarities for matching and non-matching pairs (these probabilities in turn are learned to use a Naive Bayes classifier).

- Next, the matching rules are applied to find matching tuple pairs. TAMR then runs a correlation clustering algorithm that uses the matching information to group tuples into a matching group.
- Finally, all tuples within each group are consolidated using user-defined rules to form a super tuple.



Hình 5: Key TAMR Capabilities

3. **Pros and Cons:**

   (a) **Pros:**

   - The tool supports not only single table or two tables, but also a multiple tables context.
   - It also supports a variety of data formats including XML, CSV, SQL & NoSQL databases, ...
   - Also, since it uses a web-based interface, it would be more convenient for general users to use the tool efficiently than GUIs or CLIs on non-commercial systems.

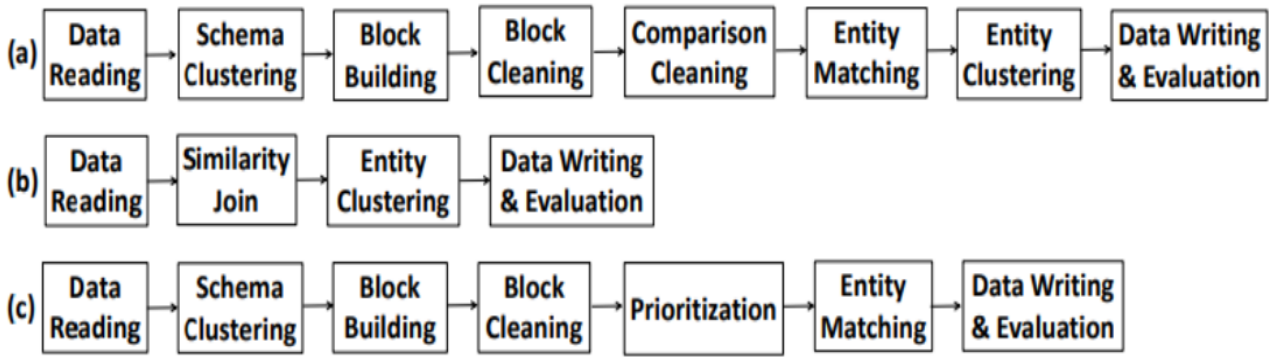
   (b) **Cons:**

   - There is still no distinction between production and development stages or any documentations on it.
   - Also, this tool uses Java as their main interacting language, so there are limited scripting environments, and it's also not supporting a lot of useful features in the data science stacks.

2.6 JedAI

Java gEneric DAta Integration Toolkit or JedAI [6] is a popular tool right now in next generation of EM systems where numerous state-of-the-art methods for all steps of the end-to-end ER workflow are implemented.

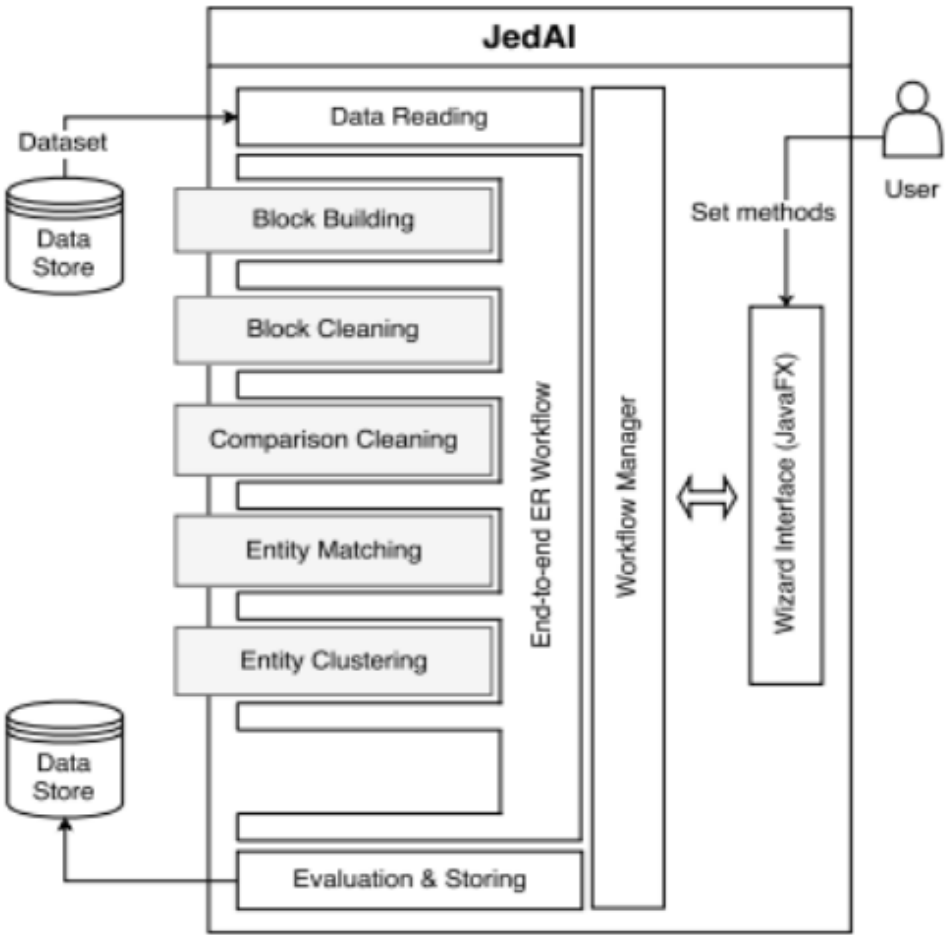
1. **Workflow:**



Hình 6: JedAI Workflows. **(a) blocking-based; (b) join-based; (c) progressive**

- *Data Reading*: loads sets of resources along with the corresponding golden standard.
- *Block Building*: receives as input the data source(s) loaded by Data Reading and clusters their resources into a set of blocks that is returned as output.
- *Block Cleaning*: receives as input the blocks produced by Block Building, discards both types of unnecessary comparisons when overlapping by enforcing constraints on the level of individual blocks.
- *Comparison Cleaning*: receives as input a set of blocks and aims to clean it from its unnecessary comparisons, just like Block Cleaning. The difference is that Comparison Cleaning operates at a finer granularity, targeting individual comparisons.
- *Entity Matching*: is a mandatory step that executes all comparisons that are contained in the set of block it receives as input and produces a similarity graph.
- *Entity Clustering*: is a mandatory step that receives as input the similarity graph of Entity Matching and partition its nodes into equivalence clusters such that every cluster contains all resources that correspond to the same real world object.
- *Evaluation & Storing*: estimates the performance of the resulting set of equivalence clusters with respect to the established effectiveness measures (Precision, Recall and F-Measure)

2. **Architecture:**



Hình 7: JedAI Architecture

JedAI has a modular architecture and there is a separate module for every step in the workflow above. Each module exposes a well-defined interface so that any method that implements it can be seamlessly integrated into that module. This makes it easy to add more ER techniques in the future, as they become available.
Additionally, every interface takes special care to ensure that the functionality of each method is well-documented and that its configuration parameters are associated with a short explanation of their role along with their possible values.
Finally, JedAI's architecture is extensible with additional modules that could add more steps to the workflow.

2.7 Four limitations of entity matching system

Based on our introduction on some of the main past solutions of building EM systems. Here are four of their main limitations.

- First, systems do not cover the entire EM pipeline. There are two main steps for entity matching. That are blocking and matching. The blocking step aims to remove obvious non-matching tuple pairs and reduce the set considered for matching based on heuristics. Entity matching in practice involves many steps than just blocking and matching. EM users have to take many steps to perform EM such as blocking, matching, exploring, cleaning, debuging, sampling, labeling, estimate accuracy, etc. However, the systems do not cover the entire EM pipeline. They only provide a limited number of steps such as blocking and matching, and overlook other essential steps such as sampling and debugging.
- Second, the majority of existing EM systems are self-contained that are not designed from the scratch to "play well" with other systems. Meanwhile, there are many techniques to be exploited by EM steps such as learning, mining, visualization, outlier detection, information extraction, etc. Additionally, integrating all of this techniques to a system is extremely difficult.
- Third, it is difficult to write code to "patch" the system. In real-world scenarios, users frequently have to write code to either add lacking functionality or to tie together system components. Obviously, it is hard to write such code correctly in "one shot". Thus such coding should be done using an interactive scripting environment, to enable rapid prototyping and iteration. However, current EM systems do not provide these facilities.
- Finally, users do not know what step to take in many EM scenarios. We can suppose a user want to perform EM with 90 percent precision and 85 percent recall. He/ she wonders which techniques can be used in here. Should he/ she use a learning-based EM approach or a rule-based approach or both? If learning-based, then which technique should be selected among the many existing ones (e.g., decision tree, SVM, etc.)? What to do if after many tries the user still cannot reach 85 percent recall with a learning-based approach? Current EM systems does not support to no answers such questions.

Looking at all these limitations, that's where Magellan comes in the picture as one of the optimal solutions to solve the main limitations from EM systems.

3 The Magellan Tool

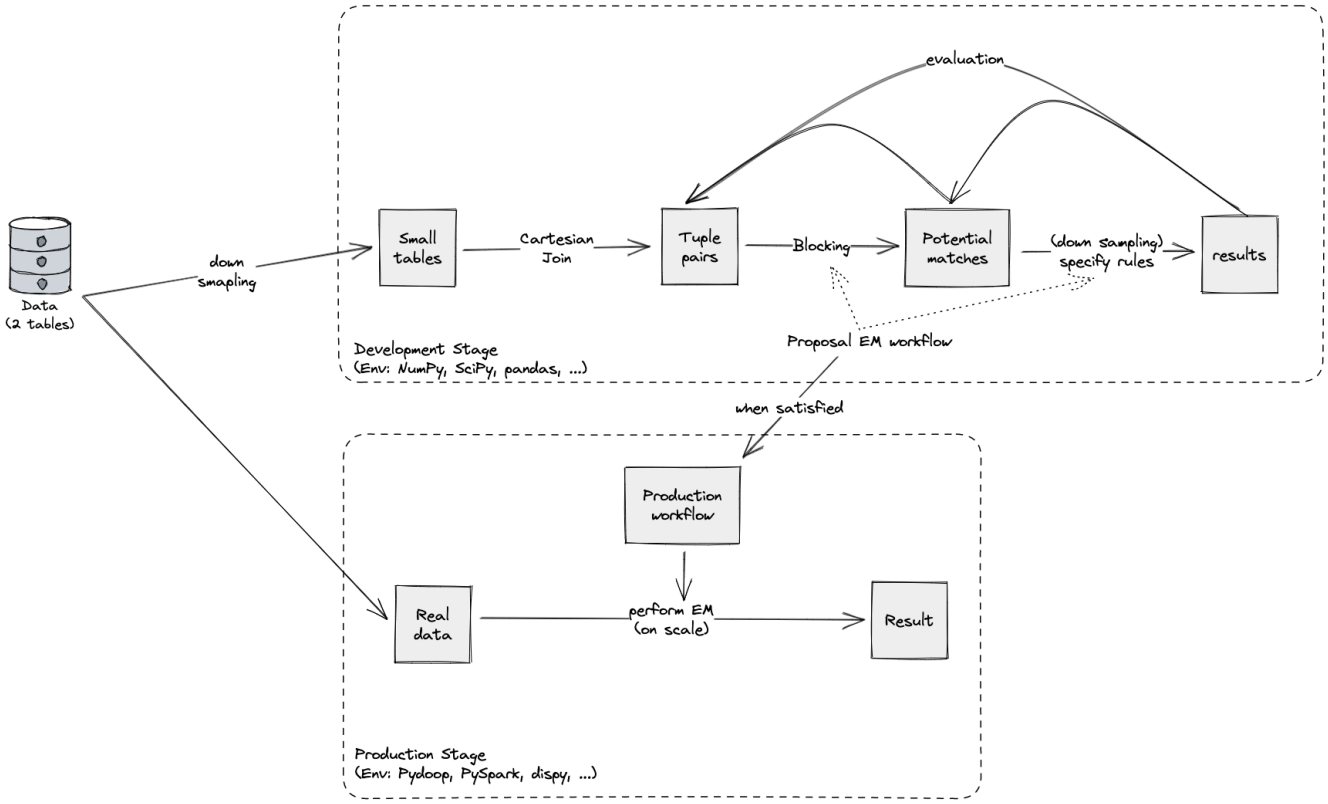
Magellan is a new kind of EM systems that was developed at UW-Madison, in collaboration with WalmartLab in 2015 [7]. However, it is not a stand-alone monolithic system such as RDBMSs or most current EM systems, but is actually an ecosystem of interoperable EM tools, and together with a recommendation for building EM systems [8]:

1. **How-to guide**: A overall proposal solution for solving the Entity Matching scenarios end to end.
2. **Tools**: Any Python libraries (`py-entitymatching, py-stringmatching, py-stringsimjoin`) or any services, created as a supporting tools for the above guide.

By separating between these two concepts, we can be easier of understanding this tool. Also, reevaluation of the solutions would be fine-grain and less subjective.

3.1 The proposal guide

Taking account of the problems arose with existing systems, the creators of the Magellan tool have proposed a how-to guide to tackle the Entity Matching problem, which can be summarized as the below diagram:



Hình 8: Magellan team proposal of Entity Matching workflow

3.2 Split of Development and Production

The authors found that in for most EM problem, the guide can be split into 2 stages: **Development** and **Production** stage.

In the **development** stage, users only operate on a small sets of data. By doing so, the users can easily test and tuning the steps of their solution (called **EM workflow**) freely and quickly, without the need for high performance machines. Specifically, to be more interactive, this stage would use lightweight but efficient libraries, like Pandas or SciPy on Python. However, problems may arise during this stage, such as: *"How to correctly sample data that represent correctly the whole sample".*

When users are satisfied with their workflow, they can move on to the **production** stage. In this stage, users then applied techniques and specialized hardware modules (such as **MapReduce** [9] [10] or parallel / distributed computing) to perform EM for the entire datasets. Nevertheless, it seems like the authors have not really found a solution for the development-to-production workflow migration problem. In the research paper, "to scale, U may need to rewrite the code for blocking and matching to use Hadoop or Spark" [7]. Bridging the gap between these two environments would bring a lot of benefits, as it would allow a faster round evaluation using real data.

The split between development and production environment is not new, at least in the Machine Learning context. Nonetheless, this is a very new concept for the Entity Matching problem. It is really promising that this optimization would bring a lot of benefits to the result.

3.3 Workflow

As mention in the previous section, the creator of Magellan propose a concept of **EM Workflows** to describe steps of performing Entity Matching. This idea is similar to the **pipeline** in an ETL process in that a workflow consists of many consecutive steps, for instance: data cleaning, **Information Extraction** (IE), blocking and matching.

Still, in the API of Magellan, the Workflow is just a conceptual entity. That is, there is no real implementation of a workflow in the library. Perhaps, the authors are just experimenting with this idea, or they want to give the users a higher degree of flexibility in exchanging the idea. Whatever it is, the concept can easily create misconceptions among newcomers.

3.4 Loading and Downsampling

First and furthermost, the important task in Development Stage is to load data into 2 separate tables. What we are dealing now is the trade-off between the volume of sample data versus number of matches. Since this stage is iterative by nature, working with large tables can be very time-consuming and frustrating to the user. Random sampling however does not work, because those 2 tables may end up sharing very few matches, yet it may not get all important matching categories into 2 new tables. Major challenges here include how to effectively cluster tuples from the large tables A and B, and how to define and infer matching categories accurately.

3.5 Blocking

In the next step, we apply blocking to the two tables to remove obviously non-matching tuple pairs. Ideally, this step should be automated as much as possible. One straightforward solution is to label a set of tuple pairs, then use that set to automatically propose a blocker. Many blocking solutions have been developed, including overlap, attribute equivalence (AE), sorted neighborhood

(SNB), hash-based, and rule-based methods. By using a debugger to judge whether a tuple pair is likely or unlikely to match, we can quickly search for such pairs in D and know when to stop tuning a blocker.

3.6 Sampling and Labeling Tuple Pairs

After applying blocking to the two tables, we obtain a set of tuple pairs that are likely to match. However, not all of these pairs are true matches, and some may be false positives. To train a matcher that can correctly identify true matches, we need to label a subset of these pairs as either matched or not matched.

This labeling process can be time-consuming and challenging, as it requires careful consideration of the attributes and context of each pair. It is important to ensure that the labeled pairs are representative of the entire dataset and cover a range of possible matching scenarios.

Overall, the Sampling and Labeling Tuple Pairs step is a critical part of the entity matching process, as it enables us to train a matcher that can accurately identify matches and improve the quality of our data.

3.7 Matcher

In the Magellan architecture, the matcher is responsible for comparing pairs of records and determining if they refer to the same entity. The matcher takes as input a set of records, a set of blocking rules, and a set of similarity functions.

The matcher first applies blocking to the records, which reduces the search space by partitioning the records into smaller blocks. The matcher then applies the similarity functions to compare pairs of records within each block.

The output of the matcher is a set of pairs of records that are identified as matches. These matches can then be used for further processing, such as entity resolution or data integration.

In the next step, the user debugs those matchers to improve its accuracy. Such debugging is critical in practice, yet has received very little attention in the research community. Authors' guide suggests that user U debug in three steps: identify and understand the matching mistakes, categorize these mistakes, and take actions to fix common categories of mistakes. Subsequently, since the data, labels, and features may have changed, U would want to do cross validation again to select a new "best matcher", and so on. This continues until you are satisfied with the accuracy of the EM workflow. Then this workflow is now onto the production stage.

3.8 PyMatcher versus CloudMatcher

PyMatcher and CloudMatcher both use the Magellan architecture, but serve different purposes. PyMatcher heavily relies on machine learning, particularly in its learning-based matcher [8] It is designed for those who have knowledge of programming, Machine Learning, and Entity Matching.

In contrast, CloudMatcher is an Entity Matching cloud service for individuals who have little to no knowledge of programming, machine learning, or Entity Matching, but understand what it means for two tuples to match [8]. These individuals can label tuple pairs as a match or no-match, which is known as crowd-sourcing.

3.9 How can Magellan solve the limitations of current EM systems?

First, Magellan provides tools that help users do EM steps. These tools seek to cover the entire EM pipeline (e.g., debugging, sampling), not just the matching and blocking steps.

Second, the tools are being built on top of the PyData (Python packages for data analysis) and Big Data stacks. The development stage basically performs data analysis. Magellan tools were built for this stage on top of the well-known Python data analysis stack, which provide a rich set of tools such as pandas, scikit-learn, matplotlib, etc. Similarly, there are tools for the production stage on top of the Python Big Data stack (e.g., Pydoop, mrjob, PySpark, etc.). Obviously, Magellan is well integrated with Python data ecosystem. This alows users to exploit many techniques such as learning, mining, visualization, IE, etc. While most current EM systems are stand-alone monoliths, Magellan is designed to be placed within an "ecosystem" and is expected to "play well" with others (Python packages,..)

Third, the integration with Python has an additional advantage that Magellan is placed within a robust interactive scripting environment, which allows users to create prototype code to "patch" the system.

Finally, Magellan provides how-to guides that tell users what to do in each EM scenarios, step by step. So users will know which EM approach they can implement or which techniques they can select among amount of existing techniques.

4 Experiments

4.1 Implement description

In this subsection, we will introduce some main Magellan API which we used to implement the above pipeline for our two specific dataset matching experiment.

1. **Down sample**:
   Tables are down sampled to smaller size. Although the size of tables is not too large, they are still down sampled to fully perform the above pipeline.

```
sample_A, sample_B = em.down_sample(A, B, size=500, y_param=1)
```

2. **Profiling data**:
   Get general information about data such as overview, variable, interaction, correlations.

```
pandas_profiling.ProfileReport(A)
```

3. **Blocking**
   Block_tables join two input tables and perform blocking on the entire joining output. A record is removed if it does meet the attribute blocking condition the user defines.

```
C = blocker.block_tables(A, B, 'zipcode', 'zipcode', l_output_attrs=['
                                        name'], r_output_attrs=['name'])
```

4. **Debuging Blocker**

   Debuging Blocker command takes in two input tables A, B, blocking output C and returns a table D containing a set of tuple pairs that are potential matches and are not present in the blocker output C base on the correspondent attributes.

```
D = em.debug_blocker(C, A, B, attr_corres=corres)
```

5. **Get sample table**

   The supervised learning-based matcher is used in this section, so we need to label the data. To create labeled data, first we need to sample of candidate set pairs and then label them.

```
S = em.sample_table(C, 100)
```

6. **Create set of feature and extract feature vectors**

```
match_t = em.get_tokenizers_for_matching()
match_s = em.get_sim_funs_for_matching()
atypes1 = em.get_attr_types(A) # don't need, if atypes1 exists from
                                    blocking step
atypes2 = em.get_attr_types(B) # don't need, if atypes2 exists from
                                    blocking step
match_c = em.get_attr_corres(A, B)
match_f = em.get_features(A, B, atypes1, atype2, match_c, match_t,
                                    match_s)
```

   Once we have created a set of features, we use them to convert labeled sample to a table of feature vectors.

```
H = em.extract_feature_vecs(G, feature_table=match_f, attrs_before=['
                                        title'], attrs_after=['
                                        gold_labels'])
```

7. **Choose best matcher**
   For choosing the best matcher in the considered matchers, we perform cross-validation to validate the precision, recall and f1 score of each matcher.

```
result = em.select_matcher(matchers=[dt, rf], table=train, exclude_attrs=[
                                        '_id', 'ltable_id', 'rtable_id'],
                                        target_attr='gold_labels', k=5)
```

8. **Training model**
   After choosing the best model, we train the matcher using the fit command on the training data.

```
dt.fit(table=H, exclude_attrs=['_id', 'ltable_id', 'rtable_id'],
                                        target_attr='gold_labels')
```

9. **Evaluate model**

   After training the model, we predict it on the test data and validate the result. If the result does not maximize the metric we want, we can come back to previous steps to improve the metric

```
pred_table = dt.predict(table=H,  exclude_attrs=['_id', 'ltable_id', '
                                        rtable_id', 'gold_labels'],  append
                                        =True, target_attr='
                                        predicted_labels')
eval_summary = em.eval_matches(pred_table, 'gold_labels', '
                                        predicted_labels')
```

10. **Production stage**

    Perform a blocking step on two original datasets, extract the feature vectors and use the chosen matcher to match all the record in the blocking output to get the final result.

4.2 Testing for benchmark datasets

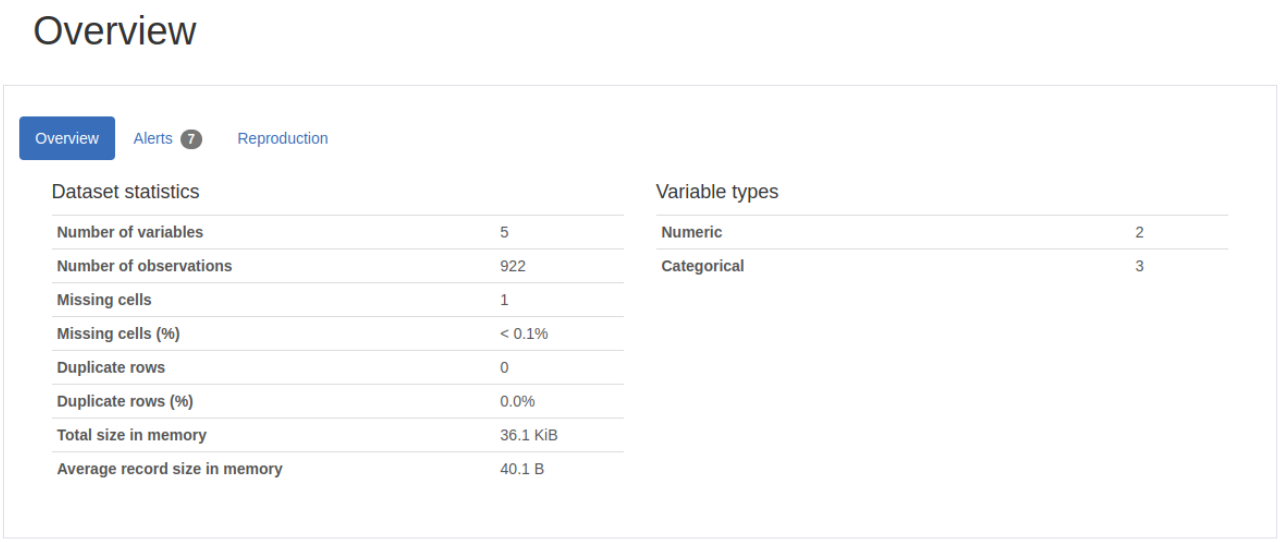
In this section, we used the ACM-DBLP dataset [11] to perform the Entity Matching task with Magellan. This data set was taken from the Benchmark datasets for entity resolution web page. It contains bibliographic data, with 4 attributes: title, authors, venue, year. There are 3 CSV files in this zip archive. All step below is implemented using API which briefly introduced at subsection 4.1

Read datasets: ACM and DBLP datasets are read into two tables. Tables in py_entitymatching are represented as pandas DataFrames. The metadata of these tables are also stored, such as key and foreign key.

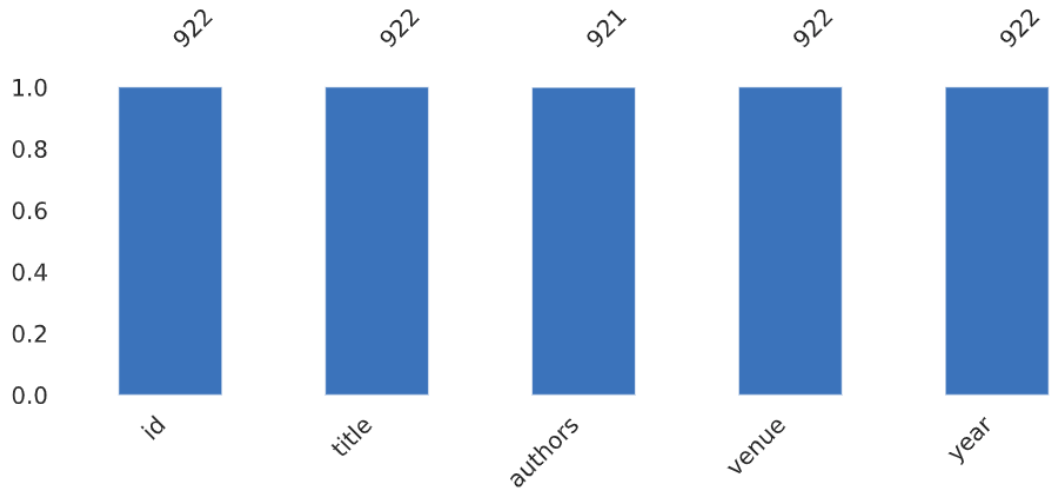
Down sampling: The *down_sample* command in Magellan samples two tables intelligently that ensures a reasonable number of matches between them. Each table then is resized approximately to 1000 records.

Profiling data: Profiling data is used to get general information about data. Below is some useful information of attributes in each table.

ACM sample:



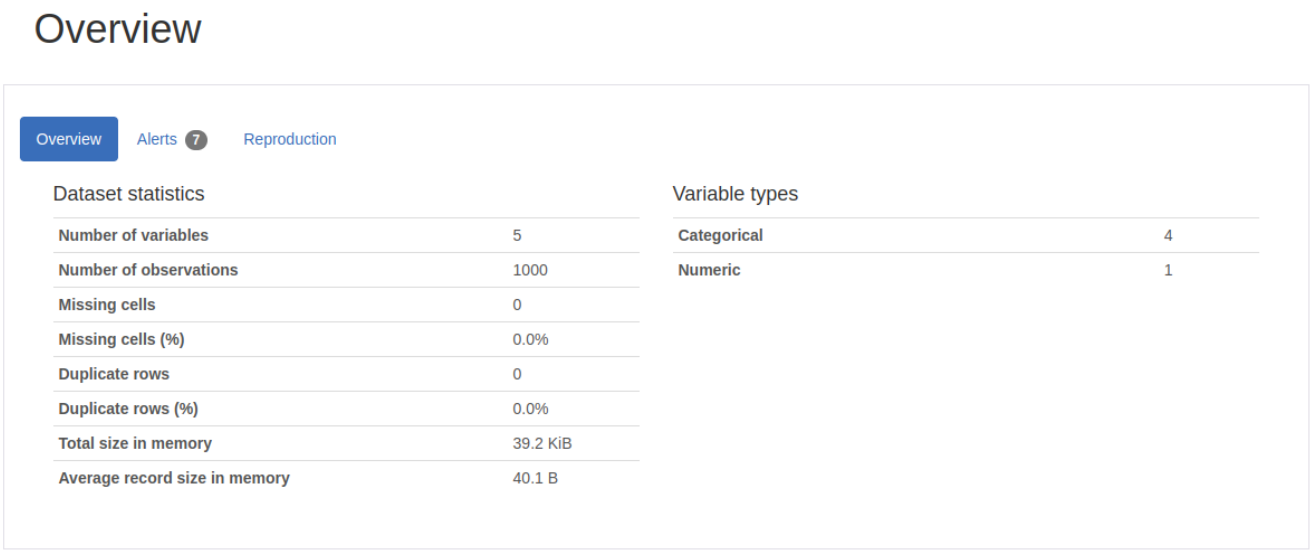
Hình 9: Overview ACM sample



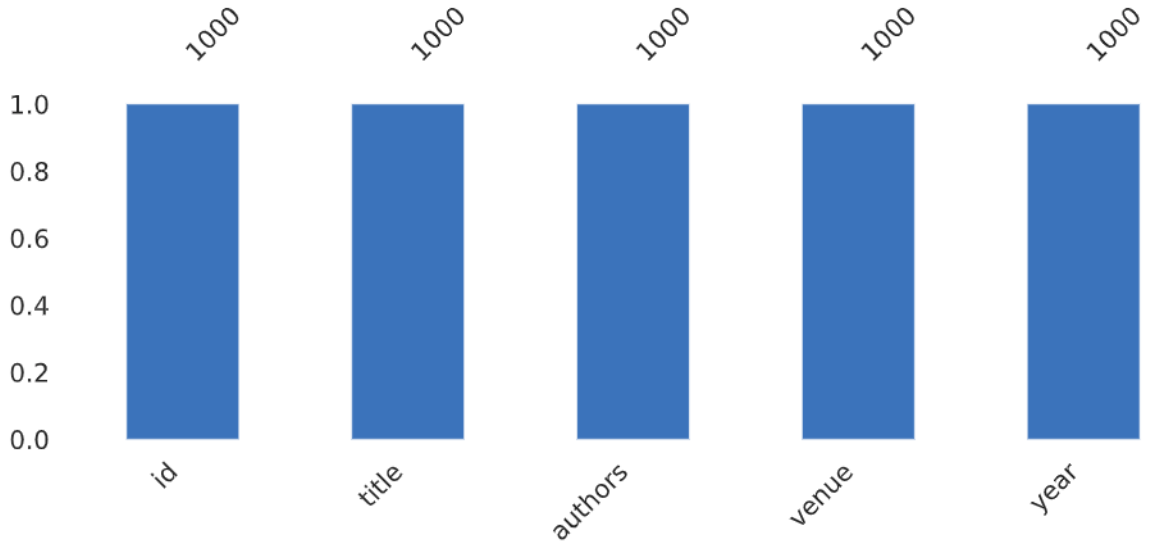
Hình 10: Visualization of nullity by collumn

From above figures, we can see the missing value in figure 9 in "authors" column of ACM sample.

DBLP sample:



Hình 11: Overview DBLP sample



Hình 12: Visualization of nullity by collumn

Blocking: There's high probability that two entities can be a match if attribute "title" and attribute "year" of each entity are match. So we decided to block these obvious attribute first. With the entities have length of title greater than three words, we use overlap blocker to the title with overlap_size=3 and equivalence blocker to the year to produce the first output. For the entities have length of title of one or two, we do the same to produce the second output, but with the overlap_size=1. If we use equivalence blocker to the "title", some special characters may be mismatched(e.g ".","-", etc.). The "venue" attribute is also considered to block records which venue is not correspondent to each other.

Combine blocker outputs: we combine two blocker output above into one table.

Debugging blocker: We debug blocker to get all entity pairs which can be a match but has been removed by blocker.

Sampling table: we get sample table size 600 in combined blocker to training our model.

Labeling for all pairs: Depend on perfect matching table, or we can use the GUI of Magellan, we label all the entity pairs with the ground truth value.

Split table: the table is then split into two table: the training set table and the test set table. The training set account for 70% of the sampled table. The matcher will use the training set for training and test set for testing the accuracy.

Extract feature vector: get all the relevant feature vectors generated based on type of four attributes and the similarity function of Magellan (e.g, Jaccard, Hamming distance, Levenshtein distance, etc.)

Choose the best matcher: test all the considered learning based matcher (random forest, Baive Bayes, SVM, decision tree, linear regression, logistic regression) using 5-fold cross validation. The matcher is chosen based on the f1 score.

Training matcher: the matcher is trained on the training set table with the chosen matcher (Naive Bayes in this report result).

Testing matcher: the matcher is then evaluated on the testing set table by predicting all the records. The report is generated with common metric like precision, recall and f1 score.

Production stage: the original dataset is used to perform matching using the tested learning based matcher. The final report is also generated.

4.3 Report result

- After blocking 2 sample dataset on ACM and DBLP, we combine 2 tables and have result:

From the figure 13 above, we can clearly see some match record which just be filtered by our blocker in line 1,2,3 and 6. Our blocker's blocking logic on attribute "title" and "year" worked pretty well.

- The debug of blocker output is shown below:

All the pairs in figure 14 are clearly unmatched, because the title, authors of left table are different from the corresponding attributes in right table.

- The extracted feature vectors depend on attribute data type of training data and have result as below :

	_id	ltable_id	rtable_id	ltable_title	ltable_authors	ltable_venue	ltable_year	rtable_title	rtable_authors	rtable_venue	rtable_year
4	4	181554	journals/sigmod/Yang94	A hypertext query language for images	Li Yang	ACM SIGMOD Record	1994	A Hypertext Query Language for Images	Li Yang	SIGMOD Record	1994
5	5	181556	journals/sigmod/GeppertD94	Constructing the next 100 database management systems: like the handyman or like the engineer?	Andreas Geppert, Klaus R. Dittrich	ACM SIGMOD Record	1994	Constructing the Next 100 Database Management Systems	Andreas Geppert, Klaus R. Dittrich	SIGMOD Record	1994
7	7	181560	journals/sigmod/JensenCEGHJ94	A consensus glossary of temporal database concepts	Curtis Dyreson, Fabio Grandi, Wolfgang Käfer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoul...	ACM SIGMOD Record	1994	A Consensus Glossary of Temporal Database Concepts	Shashi K. Gadia, Christian S. Jensen, Patrick J. Hayes, Ramez Elmasri, Sushil Jajodia, James Cli...	SIGMOD Record	1994
18	18	187455	journals/sigmod/Widom94	The impact of database research on industrial products (panel)	José A. Blakeley, Dan Fishman, David Lomet, Michael Stonebraker	ACM SIGMOD Record	1994	Research Issues in Active Database Systems: Report from the Closing Panel at RIDE-ADS '94	Jennifer Widom	SIGMOD Record	1994
20	20	187457	journals/sigmod/GeppertD94	Research issues in active database systems: report from the closing panel at RIDE-ADS '94	Jennifer Widom	ACM SIGMOD Record	1994	Constructing the Next 100 Database Management Systems	Andreas Geppert, Klaus R. Dittrich	SIGMOD Record	1994
22	22	187457	journals/sigmod/Widom94	Research issues in active database systems: report from the closing panel at RIDE-ADS '94	Jennifer Widom	ACM SIGMOD Record	1994	Research Issues in Active Database Systems: Report from the Closing Panel at RIDE-ADS '94	Jennifer Widom	SIGMOD Record	1994

Hình 13: Sample datasets to train model

	_id	ltable_id	rtable_id	ltable_title	ltable_authors	ltable_venue	rtable_title	rtable_authors	rtable_venue
0	0	202667	journals/sigmod/Date03	The third manifesto	Hugh Darwen, C. J. Date	ACM SIGMOD Record	Edgar F. Codd: a tribute and personal memoir	C. J. Date	SIGMOD Record
1	1	363956	journals/sigmod/Kim99	Theory of dependence values	Rosa Meo	ACM Transactions on Database Systems (TODS)	Message from Editor-in-Chief, ACM Transactions on Database Systems	Won Kim	SIGMOD Record
2	2	945725	journals/sigmod/Hellerstein03	Exposing undergraduate students to database system internals	Anastassia Ailamaki, Joseph M. Hellerstein	ACM SIGMOD Record	Toward network data independence	Joseph M. Hellerstein	SIGMOD Record
3	3	344819	journals/sigmod/MeltonE01	SQL standardization: the next steps	Andrew Eisenberg, Jim Melton	ACM SIGMOD Record	SQL Multimedia and Application Packages (SQL/MM)	Jim Melton, Andrew Eisenberg	SIGMOD Record
4	4	276388	conf/sigmod/HanKS97	MultiMediaMiner: a system prototype for multimedia data mining	Osmar R. Zaïane, Jiawei Han, Ze-Nian Li, Sonny H. Chee, Jenny Y. Chiang	International Conference on Management of Data	GeoMiner: A System Prototype for Spatial Data Mining	Jiawei Han, Nebojsa Stefanovic, Krzysztof Koperski	SIGMOD Conference

Hình 14: Blocker debug result

	_id	ltable_id	rtable_id	title_title_jac_qgm_3_qgm_3	title_title_cos_dlm_dc0_dlm_dc0	title_title_mel	title_title_lev_dist	title
1255	1255	959086	journals/sigmod/GalianoM03	0.148148	0.289157	0.620316	110.0	0.2
742	742	671172	conf/vldb/Moerkotte98	0.208000	0.319801	0.674752	60.0	0.2
246	246	261125	journals/tods/FranklinCL97	0.621053	0.285714	0.892638	6.0	0.9
37	37	191901	conf/sigmod/ChristophidesACS94	0.558824	0.285714	0.921654	5.0	0.9
579	579	507358	journals/sigmod/Wu02	0.038314	0.147087	0.580013	164.0	0.1
172	172	233365	conf/sigmod/HullZ96	0.575221	0.333333	0.836834	8.0	0.9
82	82	219730	journals/sigmod/DunhamH95	0.073529	0.084515	0.648158	64.0	0.2
457	457	336583	conf/sigmod/LiBCS00	0.483146	0.125000	0.866559	8.0	0.8
1108	1108	765803	journals/vldb/DeyBS96	0.529412	0.200000	0.849742	4.0	0.8
1164	1164	872823	conf/sigmod/ChengKP03	0.636364	0.333333	0.956731	4.0	0.9

Hình 15: Get feature vectors based on attribute type and similarity function

- All the considered matcher is then tested and is chosen based on f1 score. Here is the result of 5-fold cross validation

	Matcher	Average precision	Average recall	Average f1
0	DecisionTree	0.985427	0.978145	0.981689
1	RF	0.992582	0.989136	0.990781
2	SVM	0.960235	0.963435	0.961472
3	LinReg	0.996296	0.985710	0.990924
4	LogReg	0.992655	0.981444	0.986781
5	NaiveBayes	0.989209	0.992982	0.990952

Hình 16: Result of 5 model.

From this result, we can see that average f1 of Naive Bayes model is the highest, so we will use Naive Bayes to train our model.

- We use Naive Bayes to train and predict for the testing dataset.

```
Precision : 99.12% (113/114)
Recall : 99.12% (113/114)
F1 : 99.12%
False positives : 1 (out of 114 positive predictions)
False negatives : 1 (out of 66 negative predictions)
```

Hình 17: Prediction on the training set table

The precision on sample dataset is 99.12%, which show the Naive Bayes matcher is well-fitted on this dataset and is ready to apply in the production stage.

4.4 Evaluation

In this section, we evaluate the above workflow in the production stage on two original datasets (production stage). After the blocking step, we evaluate the blockers by debugging command in Magellan:

	_id	ltable_id	rtable_id	ltable_title	ltable_authors	ltable_venue	rtable_title	rtable_authors	rtable_venue
0	0	202667	journals/sigmod/Date03	The third manifesto	Hugh Darwen, C. J. Date	ACM SIGMOD Record	Edgar F. Codd: a tribute and personal memoir	C. J. Date	SIGMOD Record
1	1	363956	journals/sigmod/Kim99	Theory of dependence values	Rosa Meo	ACM Transactions on Database Systems (TODS)	Message from Editor-in-Chief, ACM Transactions on Database Systems	Won Kim	SIGMOD Record
2	2	945725	journals/sigmod/Hellerstein03	Exposing undergraduate students to database system internals	Anastassia Ailamaki, Joseph M. Hellerstein	ACM SIGMOD Record	Toward network data independence	Joseph M. Hellerstein	SIGMOD Record
3	3	344819	journals/sigmod/MeltonE01	SQL standardization: the next steps	Andrew Eisenberg, Jim Melton	ACM SIGMOD Record	SQL Multimedia and Application Packages (SQL/MM)	Jim Melton, Andrew Eisenberg	SIGMOD Record
4	4	276388	conf/sigmod/HanKS97	MultiMediaMiner: a system prototype for multimedia data mining	Osmar R. Zaïane, Jiawei Han, Ze-Nian Li, Sonny H. Chee, Jenny Y. Chiang	International Conference on Management of Data	GeoMiner: A System Prototype for Spatial Data Mining	Jiawei Han, Nebojsa Stefanovic, Krzysztof Koperski	SIGMOD Conference

Hình 18: Blocker debugging result

All the record in figure 18 is clearly unmatched, logics of the blocker is good enough in not blocking the other matched records of the original sets.

Then we extract features vectors from the output of the blocking step, the trained Naive Bayes model is used to predict the matching entity pair base on those feature vectors. The result of the final prediction is presented in figure 19

```
Precision : 95.71% (2207/2306)
Recall : 99.32% (2207/2222)
F1 : 97.48%
False positives : 99 (out of 2306 positive predictions)
False negatives : 15 (out of 2659 negative predictions)
```

Hình 19: Prediction of sample test

5 Improvement proposal

`py_entitymatching` library of Magellan is a great tool for entity matching task of data integration problem with provided pipeline. With the ACM-DBLP experimental dataset, the Magellan strategy executed flawlessly. However, we think that this tool can be better if it could resolve some of the following problems:

5.1 More sophisticated blocker and matcher

Take string blocking as an example. Currently, Magellan can only perform blocking on two texts if their tokens are similar (using edit distance, Jaccard distance, ...). This is a simple and intuitive for simple tables and entities. However, there are edge cases when similar tokens does not mean similar contents. A semantic blocker, in this context, would be a better choice. When working with the meaning of the text, not the actual symbol, it would, in theory, reduce False Negative a lot.

To use more sophisticated workflow means applying techniques from other fields of research into our problem. Deep Learning, for instance, is a possible alternative way. Some Deep Learning models

can now easily compare the semantic of any strings. If possible to integrate, Magellan can have a greater advantage compare with many alternatives.

Another problem which need to improve is the feature extraction method. In `py_entitymatching` feature vectors are extracted automatically base on the attribute name and type. Feature vectors are just extracted from a pair of attributes with the same name. It does not support extracting vectors from a pair with different attribute name. If this method could be improved and used carefully, it would make the rule based and learning based matcher more robust.

5.2 Bridging the gap between Development and Production stage

As mention in the overview section, currently, there is no efficient way to deploy a workflow implemented in a Development stage to a Production stage. In our experiment, we only use Scikit-learn to deploy our model. Scikit-learn (and also NumPy) performs excellent for data that is not too big (approximately 10 millions records). However, for real-world application where the volume of data is enormous, it is necessary to *"port"* this model onto a more capable platform. To do so, a rewrite of code is required, not also mention the compatibility checks between these libraries.

This problem should, though, not be too serious because of many existing efforts to bring Pandas and Scikit-learn libraries to scale [12] [13]. Another idea is to directly use PySpark on the development machine, by setting a standalone [14] or a local cluster [15]. Whatever the options, it should be a good practice to take this section into consideration when working with Entity Matching.

6 Source code

The source code file **Perform_Magellan.ipynb** is available at `https://github.com/TuanKietTran/research_Magellan`

References

[1] H. L. Dunn, "Record linkage," *American Journal of Public Health and the Nations Health*, vol. 36, no. 12, pp. 1412–1416, 1946.

[2] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969.

[3] H. Köpcke and E. Rahm, "Frameworks for entity matching: A comparison," 2009.

[4] S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," *Information Systems*, vol. 26, no. 8, pp. 607–633, 2001, Data Extraction,Cleaning and Reconciliation, ISSN: 0306-4379. DOI: https://doi.org/10.1016/S0306-4379(01)00042-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437901000424.

[5] P. Christen, T. Churches, *et al.*, "Febrl-freely extensible biomedical record linkage," 2002.

[6] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis, "The return of jedai: End-to-end entity resolution for structured and semi-structured data," *Proceedings of the VLDB Endowment, Vol. 11, No. 12*, vol. 11, no. 12, pp. 1950–1953, 2018.

[7] P. V. Konda, *Magellan: Toward building entity matching management systems*. The University of Wisconsin-Madison, 2018.

[8] Y. Govind, P. Konda, P. Suganthan G.C., *et al.*, "Entity matching meets data science: A progress report from the magellan project," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19, Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 389–403, ISBN: 9781450356435. DOI: 10.1145/3299869.3314042. [Online]. Available: https://doi.org/10.1145/3299869.3314042.

[9] *Mapreduce tutorial*, Mar. 2023. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

[10] *Pyspark documentation*. [Online]. Available: https://spark.apache.org/docs/latest/api/python/index.html.

[11] E. R. A. T. Hanna Köpcke, *Dblp acm dataset*, 2019. DOI: 10.3886/E109263V2.

[12] S. G, *Parallel processing in python using dask*, Jul. 2020. [Online]. Available: https://medium.com/swlh/parallel-processing-in-python-using-dask-a9a01739902a.

[13] *8.3. parallelism, resource management, and configuration*. [Online]. Available: https://scikit-learn.org/stable/computing/parallelism.html.

[14] *Spark standalone mode*. [Online]. Available: https://spark.apache.org/docs/latest/spark-standalone.html.

[15] Programmer, *Docker compose for apache spark*, Sep. 2019. [Online]. Available: https://devlistener.medium.com/docker-compose-for-developers-97e2f44a91b8.