Automating Entity Matching Model Development

Pei Wang Weiling Zheng Jiannan Wang Jian Pei
Simon Fraser University
{peiw, weiling_zheng, jnwang, jpei}@sfu.ca

***Abstract*—This paper seeks to answer one important but unexplored question for Entity Matching (EM): can we develop a good machine learning pipeline *automatically* for the EM task? If yes, to what extent the process can be automated? To answer this question, we find that a general-purpose AutoML tool cannot be directly applied to solve an EM problem, thus propose AutoML-EM, an automated model pipeline development solution tailored for EM. In reality, however, another bottleneck of EM problem is the insufficient labeled data. To mitigate this issue, active learning based solutions are widely adopted. Under this setting, we propose AutoML-EM-Active, investigating how to maximize the benefit of AutoML-EM with automatic data labeling. We provide fundamental insights into our solutions and conduct extensive experiments to examine their performance on benchmark datasets. The results suggest that AutoML-EM not only avoids human involvement in model development process but also reaches or exceeds the state-of-the-art EM performance, and AutoML-EM-Active improves the model performance under the active learning setting effectively.**

***Index Terms*—Entity Matching, AutoML, Data Integration, Active Learning**

I. Introduction

Entity matching (EM) [12] is the task of finding different records that refer to the same real-world entity. For example, consider two restaurant tables A and B in Figure 1. Although record $a1$ and record $b1$ do not match exactly, they refer to the same real-world restaurant. EM has numerous applications in 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.

EM can be viewed as a Machine Learning (ML) problem, where the goal is to build an ML model which takes a record pair as input and returns either a positive label (matching) or a negative label (non-matching). However, building an ML model that solves the problem well could take data scientists a lot of time. They need to perform manual tuning and selection at several steps, such as what features to include, how to process the features, which model to select, and how to set hyperparameters for the model. As will be shown in Section II, each of these decisions could have a big impact on the final results. There is a huge search space for data scientists to be explored in order to find the optimal ML pipeline.

Automate Model Development for EM. Recently, the ML community has put a significant amount of effort into automating ML model development, including automated feature engineering, automated model selection, and automated hyperparameter tuning (see a recent book [20] for a comprehensive survey). One crucial question but has not been answered yet

ID	Name	Address	City	Type
a1	arnie mortons of chicago	435 s. la cienega blv.	los angeles	american
a2	arts delicatessen	12224 ventura blvd.	studio city	american
a3	fenix	8358 sunset blvd. west	hollywood	american
a4	restaurant katsu	1972 n. hillhurst ave.	los angeles	asian

Table A: Restaurants in Data Source A

ID	Name	Address	City	Type
b1	arnie mortons of chicago	435 s. la cienega blvd.	los angeles	steakhouses
b2	arts deli	12224 ventura blvd.	studio city	delis
b3	fenix at the argyle	8358 sunset blvd. w.	hollywood	french (new)
b4	katsu	1972 hillhurst ave.	los feliz	japanese

Table B: Restaurants in Data Source B

Fig. 1: An EM example for restaurant data (4 matching record pairs: $(a1, b1), (a2, b2), (a3, b3), (a4, b4)$.

is *whether AutoML can be used to automatically develop an EM model which outperforms a human developed model*.

We find that a general-purpose AutoML tool cannot be directly applied to solve an EM problem because it requires the input to be feature vectors rather than raw record pairs. We explore the solution for extracting features and discuss how to use AutoML techniques effectively for EM. We call our approach AutoML-EM and compare the performance with two state-of-the-art EM solutions, Magellan [31] and DeepMatcher [28]. Our study leads to two surprising findings.

- Finding 1. AutoML-EM achieves an average improvement of **5.8%** in F1-score over human developed models on a variety of benchmark datasets. This finding is derived from the comparison with Magellan, the state-of-the-art entity matching system. Both Magellan and AutoML-EM are based on non-deep-learning models. The difference between the two systems is that Magellan keeps humans in the loop and provides detailed how-to guides to help data scientists to build and tune an EM model step by step, while AutoML-EM keeps humans out of the loop and leverages cutting-edge AutoML techniques to automatically finds an optimal ML pipeline for EM. Finding 1 suggests that we need to rethink the role of human in the EM model development stage.
- Finding 2. Non-deep-learning based EM models can achieve comparable or even better performance than deep-learning based EM models [9], [28]. This finding is derived from the comparison between AutoML-EM and DeepMatcher, a state-of-the-art deep learning based EM

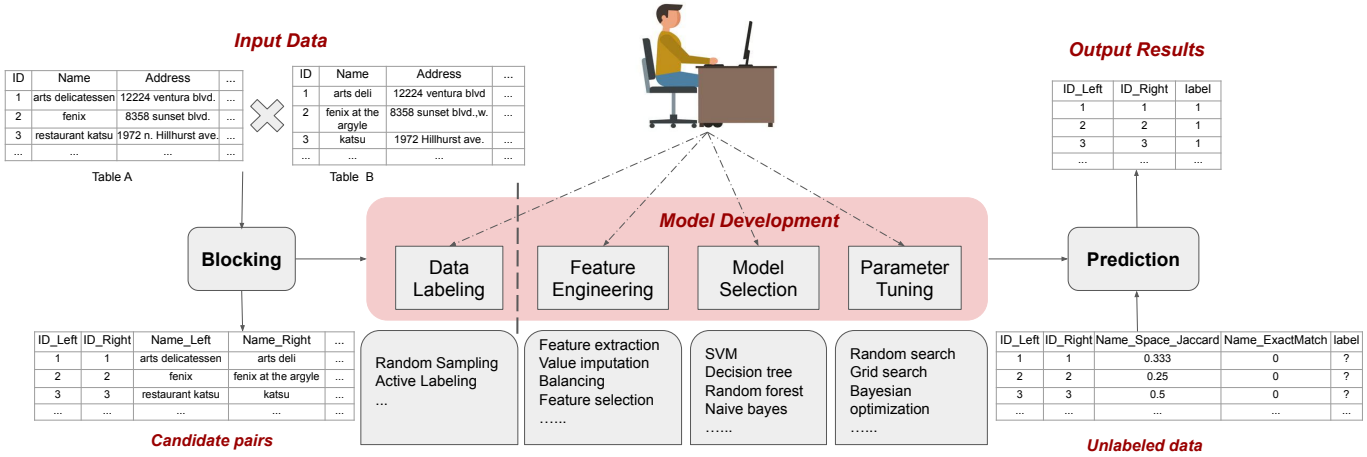


Fig. 2: An illustration of model development for entity matching.

system. As AutoML and deep learning technologies are both continuously evolving at a fast pace in the machine learning community, it is unclear how the comparison result between the two will change in the future. But at this point, data scientists should keep both of AutoML-EM and DeepMatcher in their toolbox.

Automate Data Labeling under Active Learning. Even if we know how to train a good model automatically, in practice, another labor intensive part of EM is data labeling. Data labeling has been known as a fundamental problem for decades. In response, active learning is widely adopted in the EM literature to select training examples and get effective labels [5], [6], [10], [18], [26], [27], [32], [35]. Active learning is an iterative data labeling framework. At each iteration, it uses the current model to decide which record pair to label next, asks humans to label it, and finally retrains the model on the updated training data. End-to-end EM solutions, such as Corleone [18], Magellan [31], and Dedupe [2], provide the data labeling components and the associated example selectors for effective labeling. Therefore, when building an AutoML solution for the EM problem, it is critical to consider the compatibility in the active learning setting.

A naive integration is to collect human labels via active learning and run AutoML-EM directly on the human labels. Our idea is to augment human labels with free machine-inferred labels (short as machine labels) and then run AutoML-EM on a mix of human and machine labels. The intuition is more labels often lead to better models. We leverage self-training to collect machine labels. We call this approach AutoML-EM-Active, a framework that integrates active learning, self-training, and AutoML in an EM machine learning pipeline. Self-training is a *simple* semi-supervised learning approach that gives labels for free. There are certainly other approaches that can be used to infer labels, such as transitivity [39], labeling function [8], [33], [38], clustering [11], and label propagation [43]. The purpose of our study is to show the possibility of this hybrid approach. Due to the ubiquitous use of active learning in EM, we believe that it is promising to explore how to combine active learning with other automated data labeling approaches in the future.

To conclude, this paper makes the following contributions:

- We propose AutoML-EM, an AutoML-based approach to automate model development for EM, and propose AutoML-EM-Active, a new framework that integrates the data labeling component to the AutoML-EM pipeline.
- We study in-depth how AutoML-EM helps users build an ML pipeline effectively and automatically. We validate that AutoML-EM-Active is promising to further improve EM results in the active learning setting.
- We show that AutoML-EM can automatically develop much better models than Magellan, and reach or exceed the performance of DeepMatcher on a variety of EM benchmark datasets.

The paper is organized as follows. Section II justifies why AutoML is needed for EM. Section III describes how to customize the general-purpose AutoML for EM, and Section IV discusses our AutoML-EM-Active framework. Experiments are presented in Section V, followed by related work (Section VI) and conclusion (Section VII).

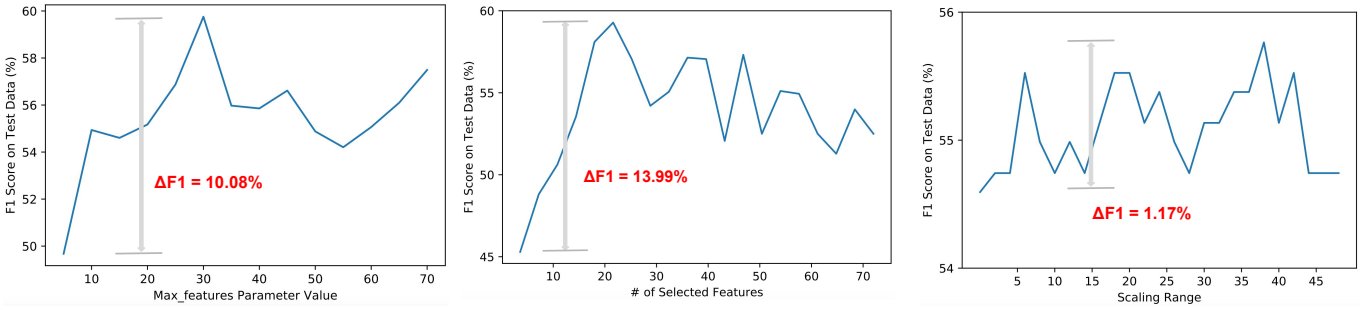
II. Why AutoML is Needed for EM?

In this section, we introduce the background knowledge of EM and then discuss the motivation in detail.

A. Entity Matching

The input of an entity matching problem is either one table or two tables. The goal is to find all pairs of records in one table or between two tables that refer to the same real-world entity [12]. Entity matching consists of two phases:

Blocking. There are $|A| \times |B|$ pairs of records to be compared potentially. To avoid the quadratic complexity of comparing all pairs, the blocking phase aims to quickly remove obviously non-matching pairs [29], and generate a small set of candidate pairs for the matching phase. One common idea is to divide data into a set of blocks and assume that the record pairs between two different blocks are non-matching and thus can



(a) Tuning Random Forest (b) Tuning Feature Selection (c) Tuning Data Scaling

Fig. 3: The effect of tuning parameters for ML pipeline components.

be safely pruned. For example, we can put the restaurants with the same city attribute into the same block, and generate the candidate restaurant pairs by enumerating every pair of records within each block.

Matching. The matching phase aims to develop the prediction model, which takes a candidate pair as input and predicts whether they are matching or non-matching. Figure 2 illustrates the model development process. Data scientists typically need to go through four steps. First, they need to construct a training dataset using active learning or random sampling. In feature engineering, they need to extract features from a record pair, perform data preprocessing like imputing missing values, handling imbalanced data, and feature preprocessing like feature selection, etc. In model selection, they need to decide which model is most suitable for her EM task. And finally, they need to tune hyper-parameters (e.g., the number of trees in a random forest, the kernel in SVM) in order to get the best model performance.

In this paper, we focus on the matching phase and treat blocking as an orthogonal problem.

Evaluation Function. The evaluation function takes an ML pipeline as input and outputs an evaluation score on the test dataset. The higher the score, the better the pipeline. F1 Score is a standard evaluation metric for EM. It is defined as the harmonic mean of Precision and Recall:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},$$

where Precision is the ratio of correctly identified matches to all identified matches and Recall is the ratio of correctly identified matches to all true matches in the data. We use F1 Score for evaluation throughout the paper.

B. Why AutoML?

Finding a good ML pipeline is challenging due to the large search space, i.e. the possible ways to assemble an ML pipeline. For example, for the feature preprocessing component alone, scikit-learn [30] provides tens of methods, where each has several parameters to tune. On the other hand, a good selection of methods and parameters can help improve prediction results. In the following, we first examine whether parameter tuning matters for EM scenario and then show that the search space is too large for humans to tune manually, which justifies the need of using AutoML for EM.

Parameter Tuning Matters for EM. We chose an EM benchmark dataset, called Abt-buy, which is a product matching dataset described in Sec V. We trained with 4/5 data, evaluated with the rest 1/5 data, and reported F1 Score. We use the same input feature vectors for the three experiments, explained in later sections.

- *Tuning Random Forest.* We trained a random forest model using Scikit-learn [30]. We set other parameters to the default value and tune the $max\_features$ parameter, which represents the maximum number of features to check when searching for the best split. If it is set a large (small) value, the model will have an overfitting (underfitting) issue. We varied $max\_features$ from 5 to 70, and computed the F1 Score of the model w.r,t, each max_features value. Figure 3a shows the result. We can see that $max\_features$ affects the model performance a lot. The F1 Score difference between the best $max\_features$ and the worst $max\_features$ is 10.08%.
- *Tuning Feature Selection.* We next examine how feature selection may affect model prediction accuracy. We adopted the `SelectPercentile` function in scikit-learn for tuning. This function ranks features based on the ANOVA F-value score and selects the top features with the highest scores. We varied the number of selected features from 5 to 70, and computed the F1 Score of the random forest model using default settings. The result is shown in Figure 3b. Like the previous experiment, we can see that the number of selected features had a big impact on the F1 Score, resulting in a F1 Score gap as high as 13.99%.
- *Tuning Feature Scaling.* Standardization, which rescales features with statistics, is a common data preprocessing step in machine learning. We chose the `RobustScaler` function [3] in scikit-learn to scale data since it is robust to outliers. It has a parameter called q_min, representing the first quantile in the interquartile range. Having a different q_min value leads to a different distribution of the rescaled feature. We varied q_min for `RobustScaler` from 0 to 50, and trained a random forest with rescaled features. The result is shown in Figure 3c. While the impact of this parameter is not as big as the previous two, it can still lead to ΔF1 Score $= 1.17\%$.

In practice, data scientists have to tune the model manually

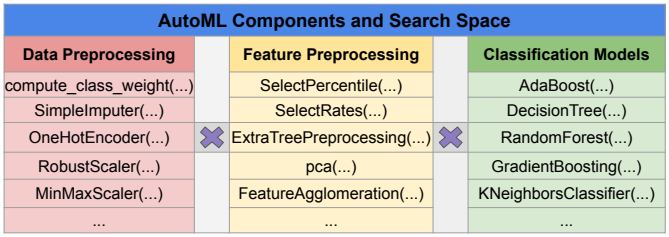


Fig. 4: Search space of AutoML.

in order to achieve the best EM model.

Search Space is Too Large to Tune Manually. It is not hard to understand that it is impossible to find out the optimal pipeline manually given the huge number of possible pipelines (i.e., the search space). Figure 3 only shows three components with three parameters. It has already had a big search space, i.e., $|max\_features|*|k|*|q\_min| = 70*70*50 = 245000$. If we consider all the components in a typical AutoML system as shown in Figure 4, the search space will get much bigger. For example, there are nearly twenty functions in the single feature preprocessing module. There are over ten different models in the model selection module. Each model can be associated with more than ten parameters. Tuning these parameters not only requires high expertise for ML models, but also careful examination through the model documentation and setting up a parameter tuning algorithm like random search.

III. Enabling AutoML for EM

In this section, we present how to enable AutoML for the EM problem. We starts with an introduction of the AutoML process, and then discuss how to effectively extend AutoML to solve the EM problem specifically. We call our approach AutoML-EM.

A. AutoML Process

The key idea of AutoML is to treat model development as a search problem. Given a training dataset and a validation dataset, AutoML aims to automatically find an ML pipeline that is executed on the training dataset and produces a model that achieves the highest performance on the validation dataset. To achieve this, AutoML first constructs a search space that contains a large number of *ML pipelines*, then defines an evaluation metric (e.g., F1 Score) to measure how good an ML pipeline is, and finally applies a search algorithm to find the best ML pipeline automatically.

Search Space. An ML pipeline is a set of steps of how to process the data and train the model. For example, auto-sklearn [1], [15] defines an ML pipeline as a sequence of four parts: data preprocessing $\rightarrow$ feature preprocessing $\rightarrow$ model selection $\rightarrow$ associated hyperparameter settings for the selected methods. Figure 5 shows a toy auto-sklearn pipeline example. We can see that two functions are applied in data preprocessing, where the associated parameters are 'weighting' and 'mean', respectively.

Search Algorithm. Given a search space, an evaluation metric, and a time budget, the goal of the search process is to find the best pipeline (e.g., with the highest F1 Score on the validation set) in the search space within the time budget.

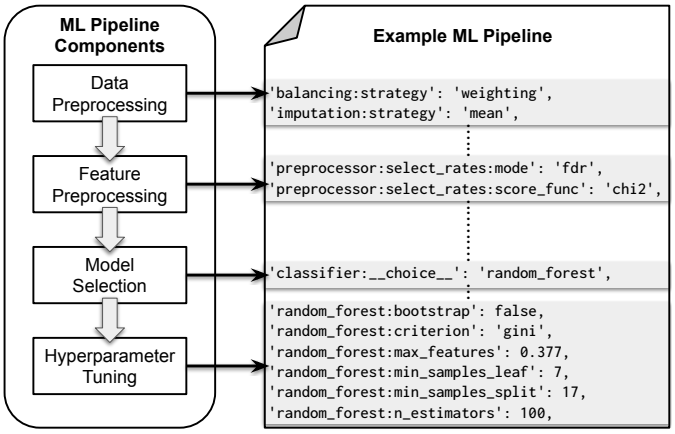


Fig. 5: An example ML pipeline generated by auto-sklearn.

The pipeline searching is the most challenging part of AutoML. A lot of research efforts have been devoted to solving this problem [7], [20], [40]. TPE [7] and SMAC [19] are the state-of-the-art search algorithms. Their basic idea is to iteratively build a *surrogate model* and use it to guide the search process. For example, SMAC builds a random forest to predict the F1 Score for a given ML pipeline. At each iteration, it uses the random forest model to predict the F1 Scores for a sample of pipelines and selects the most promising pipeline (i.e., the one with the highest expected F1 Score). After that, it executes the selected pipeline to get the actual F1 Score and updates the surrogate model accordingly. This process will be repeated until the time is used up.

B. AutoML-EM Feature Generation

We find that existing AutoML tools cannot be directly applied to EM because they require the input to be numerical feature vectors rather than textual record pairs.

Let r denote a record with m attributes $A_1, A_2, \cdots, A_m$. Let $r[A_i]$ denote the value of the record on attribute A_i. Given a record pair (r_1, r_2), the goal is to convert it to a numerical feature vector $[f_1, f_2, \cdots, f_n]$[1]. Magellan proposes an idea to solve the problem (Table I). In the following, we will explain how it works, then identify the issue of using it for AutoML, and finally present our approach AutoML-EM.

Similarity Function. A similarity function quantifies the similarity between two strings (or numbers or boolean values). There are two types of similarity functions.

A *token-based* similarity function, denoted by (simfunc, tokenizer), applies a tokenizer to split each string into a token set and computes the similarity between the two token sets. For example, consider a similarity function (Jaccard, space). To compute the similarity between "new york" and "new york city", we split each string by space and get two token sets, $T_1 = \{\text{new}, \text{york}\}$ and $T_2 = \{\text{new}, \text{york}, \text{city}\}$. We then compute their jaccard similarity as $\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{2}{3}$.

A *non-token-based* similarity function, denoted by (simfunc, N/A), directly computes the similarity between two strings,

[1]Similarity joins convert each record (rather than each record pair) to a vector and then compute vector similarity to find matching record pairs. However, as shown in prior work, they are not as effective as model-based methods [22].

ID	Data Type	Similarity Function
1	Single-Word String	(Levenshtein Distance, N/A)
2		(Levenshtein Similarity, N/A)
3		(Jaro Distance, N/A)
4		(Exact Match, N/A)
5		(Jaro-Winkler Distance, N/A)
6		(Jaccard Similarity, 3-gram)
7	1-to-5-Word String	(Levenshtein Distance, N/A)
8		(Levenshtein Similarity, N/A)
9		(Needleman-Wunsch Algorithm, N/A)
10		(Smith-Waterman Algorithm, N/A)
11		(Monge-Elkan Algorithm, N/A)
12		(Cosine Similarity, Space)
13		(Jaccard Similarity, Space)
14		(Jaccard Similarity, 3-gram)
15	5-to-10-Word String	(Levenshtein Distance, N/A)
16		(Levenshtein Similarity, N/A)
17		(Monge-Elkan Algorithm, N/A)
18		(Cosine Similarity, Space)
19		(Jaccard Similarity, 3-gram)
20	Long String (>10 words)	(Cosine Similarity, Space)
21		(Jaccard Similarity, 3-gram)
22	Numeric	(Levenshtein Distance, N/A)
23		(Levenshtein Similarity, N/A)
24		(Exact Match, N/A)
25		(Absolute Norm, N/A)
26	Boolean	(ExactMatch, N/A)

TABLE I: Magellan feature generation process.

thus it does not need a tokenizer. For example, consider a similarity function (Levenshtein Distance, N/A). The Levenshtein distance between "new yrk" and "new york" is 1 since it needs at least 1 edit (insertion, deletion, or substitution) to transform from "new yrk" to "new york".

Magellan Feature Generation. Now we present how the Magellan feature generation works. Magellan defines six data types (see Table I). For example, an attribute is 1-to-5-Word String if the average number of words of each string in that attribute is within the range of $(1, 5]$. Given a record pair (r_1, r_2), for each attribute A_i ($i \in [1, m]$), Magellan first checks A_i's data type and then gets a list of corresponding similarity functions from Table I. Using each similarity function, Magellan computes the similarity value between $r_1[A_i]$ and $r_2[A_i]$. For example, if A_i is a 1-to-5-Word String data type, there will be 8 corresponding similarity functions to this data type, so in total, 8 features will be generated w.r.t. A_i for (r_1, r_2).

Limitations of Rule-based Feature Selection. Magellan predefines heuristic rules to generate features. However, these rules have two limitations.

First, they choose similarity functions based on the average number of words. For example, if an attribute has the average number of words larger than 10, it will be marked as Long String and can only use (Cosine Similarity, Space) and (Jaccard Similarity, 3-gram) to generate two features. While these two similarity functions are suitable for long strings, there could be some short strings in the attribute for which other similarity functions (e.g., (Levenshtein Distance, N/A)) are more suitable.

Second, these cut-off points are *not* adaptive to data context. That is, no matter which dataset is given and how dirty it is, the cut-off points are always the same. For example, maybe for one dataset, the optimal cut-off should be (Single-Word, 1-to-5-Word, 5-to-10-Word, $>$ 10 words), but for another, the optimal cutoff could be totally different, e.g., (1-to-2-Word, 2-to-8-Word, 8-to-15-Word, $>$ 15 words).

ID	Data Type	Similarity Function
1	String	(Levenshtein Distance, N/A)
2		(Levenshtein Similarity, N/A)
3		(Jaro Distance, N/A)
4		(Exact Match, N/A)
5		(Jaro-Winkler Distance, N/A)
6		(Needleman-Wunsch Algorithm, N/A)
7		(Smith-Waterman Algorithm, N/A)
8		(Monge-Elkan Algorithm, N/A)
9		(Overlap Coefficient, Space)
10		(Dice Similarity, Space)
11		(Cosine Similarity, Space)
12		(Jaccard Similarity, Space)
13		(Overlap Coefficient, 3-gram)
14		(Dice Similarity, 3-gram)
15		(Cosine Similarity, 3-gram)
16		(Jaccard Similarity, 3-gram)
17	Number	(Levenshtein Distance, N/A)
18		(Levenshtein Similarity, N/A)
19		(Exact Match, N/A)
20		(Absolute Norm, N/A)
21	Bool	(ExactMatch, N/A)

TABLE II: AutoML-EM feature generation process.

AutoML-EM Feature Generation. To overcome these limitations, we remove these pre-defined rules from our feature generation process. Our philosophy is to generate as many as possible features and then delegate the feature processing part to AutoML. Table II shows how the AutoML-EM feature generation process works. For Number and Bool, AutoML-EM generates the same set of features as Magellan, but for String, AutoML-EM always uses all of Magellan's similarity functions to generate features regardless of string length.

For example, given a record pair (r_1, r_2) with four attributes, suppose their data types are Single-Word String, Single-Word String, Long String, Long String. Magellan generates $6 + 6 + 2 + 2 = 14$ features, while AutoML-EM generates $16 + 16 + 16 + 16 = 64$ features. In the experiments, we run AutoML on the feature vectors generated by AutoML-EM and Magellan, respectively, and the results show that our approach performs much better.

C. AutoML-EM Model Selection

In terms of model selection, Magellan provides several popular models at hand such as decision tree, random forest, SVM, and logistic regression, and allows users to train all these models simultaneously with the *default* hyperparameter settings. The user can compare the evaluation score of each model on the validation dataset and select the model with the highest evaluation score. The user needs to tune the parameters manually. However, hyperparameter tuning often has a big impact on model performance as we showed. The model that performs the best with the default hyperparameter setting may not perform the best after hyperparameter tuning.

In AutoML, model selection is coupled with hyperparameter tuning. It aims to select the model that performs the best with the *optimal* hyperparameter setting. A general-purpose AutoML tool's model repository includes tens of models, which are included in the pipeline search space by default.

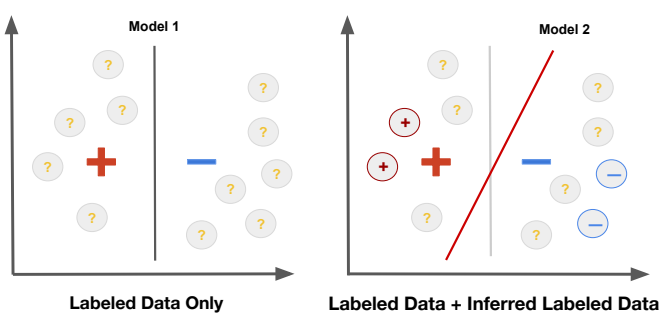


Fig. 6: An illustration of self-training.

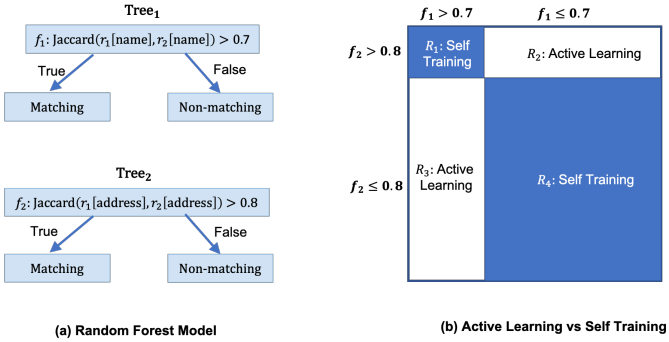


Fig. 7: An illustration of active learning vs self-training using a random forest model.

AutoML-EM is motivated by the consideration that not all the available models are necessary to be equipped for solving the EM problem. The more unnecessary models we include, the more deceleration of the optimization speed. Should we only include a few promising models to shrink search space and accelerate convergence or include as many models as possible for the best possible performance?

In our AutoML-EM solution, we only include the random forest in the model repository. The reason is that random forest has been consistently observed as the most performing one for structured data [13]. We conduct an experimental study to understand the effectiveness of this idea under different settings. So far, we have customized an AutoML solution for EM.

IV. Enabling Data Labeling for AutoML-EM

In this section, we first illustrate why it is a good idea to combine self-training and active learning for random forest model, and then describe how AutoML-EM-Active works.

Self-Training. As mentioned previously, we consider how to make the AutoML solution works better under active learning and integrate the data labeling to the AutoML-EM pipeline as well. Our idea is to get free labels with self-training. Self-training is a semi-supervised learning algorithm. It leverages both labeled and unlabeled data to train a model. The following shows its procedures:

1) Train a model on labeled data
2) Use the model to predict unlabeled data
3) Add a sample of unlabeled data with high confidence to the labeled data
4) Retrain a model on the new labeled data

Figure 6 illustrates an example. The left sub-figure shows the initial model built using labeled data (a positive point and a negative point). Self-training uses the model to predict each unlabeled point. The right sub-figure shows that self-training selects two new positive/negative points, respectively. Unlike active learning which selects the points close to the decision boundary, the points selected by self-training are far away from the decision boundary to ensure high confidence. A new model is retrained using a mix of two given labeled points and four newly inferred labeled points.

Active Learning vs Self-Training. A random forest model consists of a collection of decision trees, where each tree is trained using a sample of training data and a sample of features. To make a prediction on an unlabeled pair, the random forest model first obtains the prediction of each decision tree and then combines them (e.g., using majority vote) to get the final prediction.

Algorithm 1: AutoML-EM-Active

Input: Unlabeled record pairs U, labeling budget B
Output: A model M

T = Initial training data randomly drawn from U;
Ask humans to label T and set $b = |T|$;
Remove T from U;
M = Train a model on T;
while $b \leq B$ *and* $U \neq \phi$ **do**
  Apply M to each record pair in U and get the label confidence score of each pair;
  ac_batch = Select a batch of record pairs from U with the lowest label confidence scores;
  Ask humans to label ac_batch and set $b +=$ |ac_batch|;
  st_batch = Select a batch of record pairs from U with the highest label confidence scores and trust their predicted labels;
  Add ac_batch and st_batch to T;
  Remove ac_batch and st_batch from U;
  M = Retrain a model on T;
return *AutoML-EM model trained with collected labels*

Figure 7(a) illustrates a simple random forest model, which consists of two decision trees, Tree_1 and Tree_2. Given a record pair, Tree_1 (Tree_2) predicts it as matching if the jaccard similarity of their name (address) attribute values is larger than 0.7 (0.8); otherwise, it is non-matching.

As shown in Figure 7(b), Tree_1 and Tree_2 divide all possible record pairs into four regions. For each record pair,

- if it falls into R_1 ($f_1 > 0.7$ and $f_2 > 0.8$), then Tree_1 and Tree_2 both predict it as matching;
- if it falls into R_2 ($f_1 \leq 0.7$ and $f_2 > 0.8$), then Tree_1 (Tree_2) predicts it as matching (non-matching);
- if it falls into R_3 ($f_1 > 0.7$ and $f_2 \leq 0.8$), then Tree_1 (Tree_2) predicts it as non-matching (matching);
- if it falls into R_4 ($f_1 < 0.7$ and $f_2 < 0.8$), then Tree_1 and Tree_2 both predict is as non-matching.

R_1 and R_4 are high-confidence regions since Tree_1 and Tree_2 make consistent decisions while R_2 and R_3 are low-confidence regions since the decisions made by Tree_1 and Tree_2 are inconsistent. Therefore, self-training selects unlabeled record pairs from $R_1 \cup R_4$, while active learning selects unlabeled record pairs from $R_2 \cup R_3$.

Type	Dataset	Training Size	Test Size	# Attr.
Easy & Small	BeerAdvo-RateBeer	359	91	4
	Fodors-Zagats	757	189	6
	iTunes-Amazon	430	109	8
Easy & Large	DBLP-ACM	9890	2473	4
	DBLP-Scholar	22965	5742	4
Hard & Large	Amazon-Google	9167	2293	3
	Walmart-Amazon	8193	2049	5
	Abt-Buy	7659	1916	3

TABLE III: EM datasets.

AutoML-EM-Active. Algorithm 1 shows the pseudo-code of our hybrid active learning and self-training approach. Initially, AutoML-EM-Active asks humans to label a sample of record pairs and trains an initial model. Then, it gets into the iteration stage. At each iteration, AutoML-EM-Active applies the current model to the unlabeled data and gets the label confidence score of each unlabeled record pair. The larger the score, the more confident the inferred label. Let |ac_batch| (|st_batch|) denote the batch size of active learning (self-training), respectively. Like active learning, AutoML-EM-Active selects a set of |ac_batch| unlabeled record pairs that have the lowest confidence scores and asks humans to label them. Like self-training, AutoML-EM-Active selects a set of |st_batch| unlabeled record pairs that have the highest confidence scores and add these record pairs along with their inferred labels to the training data. AutoML-EM-Active retrains a model using the new training data. This iteration process repeats until the budget is exhausted or the unlabeled data is empty.

Remarks. (1) AutoML-EM-Active is more general than active learning. This is because active learning can be seen as a special case of AutoML-EM-Active when the self-training batch size is set to zero, i.e., $|\text{st\_batch}| = 0$. (2) Let α denote the percentage of the number of matching pairs in the initial training data T. To avoid the concept drift issue [17], we need to ensure that α keeps roughly the same between before and after adding st_batch to T. This can be achieved by choosing $\alpha \cdot |\text{st\_batch}|$ matching pairs and $(1-\alpha) \cdot |\text{st\_batch}|$ non-matching pairs at each iteration of self-training.

V. EXPERIMENTS

We conduct extensive experiments on real-world benchmark EM datasets to evaluate the performance of AutoML-EM. In the following, we first describe the datasets and settings in Section V-A, followed by an end-to-end performance evaluation in Section V-B and a detailed performance analysis of AutoML-EM in Section V-C. In the end, we investigate whether AutoML-EM-Active outperforms the case where the self-labeling process is absent in Section V-D.

A. Datasets and Settings

Datasets. All of the benchmark datasets we use have been evaluated in previous work [23], [28]. We categorize them into three types: easy and small, easy and large, hard and large. Following the setting of [28], we further split the training set to 4:1, using the 80% for training and the rest 20% for validation. Table III shows a summary of the datasets.

- BeerAdvo-RateBeer is a beer dataset with 450 record pairs (68 positive) and four attributes: beer name, brew factory name, style, ABV (Alcohol by Volume).
- Fodors-Zagats is a restaurant dataset with 946 record pairs (110 positive) and six attributes: name, address, city, phone, type, and category code.
- iTunes-Amazon dataset includes 539 record pairs of songs (132 positive) with song name, artist name, album name, genre, price, copyright, time, released attributes.
- DBLP-ACM is a publication dataset with 12363 record pairs (2220 positive), including paper title, author and venue attributes.
- DBLP-Scholar is also a publication dataset with 28707 record pairs (5347 positive) and attributes: title, authors, venue, and year.
- Amazon-Google is a software product dataset with 11460 record pairs (1167 positive) and three attributes: product title, manufacturer, and price.
- Walmart-Amazon is an electronic product dataset with 10242 record (962 positive) and product name, category, brand, model number, and price attributes.
- Abt-Buy is also a product dataset with 9575 record pairs (1028 positive) and three attributes: product name, price, and long text descriptions.

AutoML-EM is our AutoML solution customized for EM. It is equipped with our feature generation approach (see Table II) and random forest as the selected model. AutoML-EM is built upon an existing general-purpose AutoML tool, which is capable of searching for the optimized ML pipeline given a search space. In our experiments, we use auto-sklearn [1], [16], a popular open-sourced automated machine learning python toolkit, as our default tool and for evaluating our techniques. There are alternative tools written in other programming languages or based on other algorithms, such as Auto-Weka [36], TPOT [4]. A data scientist can easily tailor these tools for AutoML-EM configurations.

Experimental Settings. Our experiments were run on a Ubuntu 16.04 server with Intel Xeon E7-4830 v4 (2.00GHz) CPU and 960GB memory. For the training process, except otherwise stated, we set the default running time of auto-sklearn to 3600s. We use one hold-out method for model validation. We choose F1 Score as the evaluation metric as defined in Section II-B.

B. End-to-end Performance

We compare the end-to-end performance of AutoML-EM with the state-of-the-art EM approaches.

Magellan [31] is the state-of-the-art learning-based end-to-end EM system on structured data. It keeps the developer in the loop in every step and provides guidance for developing an end-to-end EM solution, including blocking algorithms, feature manipulation, and matching models. As we focus on

Dataset	Magellan	AutoML-EM	Δ F1 Score
BeerAdvo-RateBeer	78.8	82.3	+3.5
Fodors-Zagats	100	100	+0
iTunes-Amazon	91.2	96.3	+5.1
DBLP-ACM	98.4	98.4	+0
DBLP-Scholar	92.3	94.6	+2.3
Amazon-Google	49.1	66.4	+17.3
Walmart-Amazon	71.9	78.5	+6.6
Abt-Buy	43.6	59.2	+5.3
Average	**78.1**	**83.9**	**+5.8**

TABLE IV: An end-to-end comparison between Magellan and AutoML-EM. The performance numbers of Magellan are copied from [28].

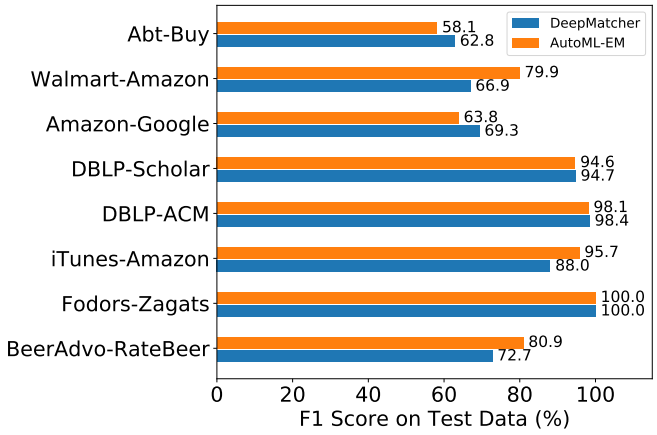


Fig. 8: Comparison of AutoML-EM with DeepMatcher. The performance numbers of DeepMatcher are copied from [28].

the matching step, we assume that the blocking step is the same for all methods and chose the matching benchmark datasets to compare the matching components of different methods.

DeepMatcher [28] is the state-of-the-art deep learning-based solution customized for EM problem. It processes the text content with NLP techniques like word embeddings and summarizations, and trains RNN model for prediction. Due to the use of deep learning, it performs better on textual data.

We compared the F1 Scores of AutoML-EM, Magellan, and DeepMatcher on eight EM datasets. Our goal is to answer the following two questions.

Can AutoML-EM beat human? To answer this question, we compare the models generated by AutoML-EM with the ones manually developed by Magellan. Table IV shows the result. Note that the F1 Scores of Magellan was copied from [28]. We can see that AutoML-EM improved human-developed models by an average F1 Score of 5.7%. On some datasets, the performance gain is quite significant. For example, the ΔF1 Scores between AutoML-EM and Magellan on iTunes-Amazon, Amazon-Google, Abt-Buy datasets are 8.8%, 17.3%, and 14%, respectively. The gain mainly comes from automated feature engineering and hyper-parameter tuning. These results validated the point that AutoML-EM can automatically develop a much better model than the model developed by humans using Magellan.

Can AutoML-EM beat deep learning? To answer this question, we compare the models automatically generated by AutoML-EM with the deep learning models generated by DeepMatcher. Figure 8 shows the result. Note that the F1 Scores of DeepMatcher were copied from [28]. We can see that AutoML-EM outperformed or was competitive with DeepMatcher on datasets like BeerAdvo-RateBee, DBLP-ACM, DBLP-Scholar, Fodors-Zagats, Walmart-Amazon, and iTunes-Amazon. These results were consistent with the results in the previous paper [28] that traditional machine learning models perform better on structured data.

A surprising new finding is that AutoML-EM (with random forest models) achieved comparable performance with DeepMatcher (with deep learning models) on text data. For example, Amazon-Google and Abt-Buy are product datasets that contain very long text description attributes. Unlike Magellan which performed much worse on these two datasets, we can see that AutoML-EM performed only slightly worse than DeepMatcher even if DeepMatcher used complex NLP models and captured semantic synonyms. It is worth noting that AutoML-EM had this major improvement over Magellan on textual data because it produced more features for long strings (please compare Table I and Table II) and then leverage AutoML techniques to process and select features.

Takeaways. *AutoML-EM can beat human-developed models by an average F1 Score of 5.7% and reach or exceed the performance of deep learning models not only on structured data but also on textual data.*

C. AutoML-EM Analysis

In this section, we decompose and analyze the benefits of AutoML-EM by answering three questions:

- Does the AutoML-EM feature generation approach lead to better results? (§V-C1)
- Does AutoML-EM with the random forest model selected produce comparable results with all models selected? (§V-C2)
- In the resulting pipeline of AutoML-EM, does module, such as data preprocessing and feature preprocessing, contribute to the improved models? (§V-C3)

1) Effectiveness of Feature Vector Generation: We compared our feature generation process against the features generated by Magellan library, i.e., Table I vs Table II. Specifically, we leveraged Magellan and AutoML-EM to convert record pairs to feature vectors, respectively, and then ran AutoML (no model selection) on both feature-vector tables and compared the results.

Figure 9 shows the number of generated features, F1 Score on test data, and the improvement achieved by the AutoML-EM feature generation approach. We can see that the AutoML-EM feature generation approach performed better on all the datasets. The reason is that the AutoML-EM feature generation approach conveyed more information and its AutoML

Dataset	Magellan		AutoML-EM		ΔF1 Score
	# Feature	Fscore	#Feature	Fscore	
BeerAdvo-RateBeer	36	81.3	87	82.3	+1.0
Fodors-Zagats	37	100	123	100	+0
iTunes-Amazon	30	88.1	155	96.3	+8.2
DBLP-ACM	18	98.3	89	98.4	+0.1
DBLP-Scholar	18	92.6	89	94.6	+2.0
Amazon-Google	21	62.9	72	66.4	+3.5
Walmart-Amazon	32	66.2	106	78.5	+2.3
Abt-Buy	15	48.1	72	59.2	+11.1

Fig. 9: Comparing the F1 Scores of AutoML with Magellan vs AutoML-EM feature generation methods.

component can automatically select good features and process them in a reasonable way. It is worth noting that Magellan's performance got improved after applying AutoML by comparing with Table IV. Thus, there is no need to use Magellan's manually pre-defined rules for these datasets.

Takeaways. *It is recommended to use all similarity functions (rather than manually select them based on string length) to generate feature vectors.*

2) Effectiveness of Model Selection: Our second idea of optimizing a general-purpose AutoML tool for EM is to shrink the model selection space. We hope to accelerate convergence speed and consequently allow the search algorithm to find a good ML pipeline in a shorter time. For comparison, we run two methods. i) all-model represents AutoML with the original model repository provided by the library. Auto-sklearn builds in tens of common models such as AdaBoost, Naive Bayes, and Decision Trees. ii) Random Forest represents AutoML with only the random forest model selected.

Figure 10 shows how both validation and test scores changed as the time constraint increased. To compare all-model and random forest, our main focus is on the validation score because this is what AutoML tries to optimize. The test score is shown for reference purpose only.

We have the following observations. 1) For all the datasets and both methods, the longer the running time, the better the validation score (there were small variations due to the randomness of auto-sklearn over different trials). For example, for the Amazon-Google dataset, AutoML-EM only has an F1 Score of 56.1% when running for 60s, but achieved 62% with 2400s running time. 2) random forest converged faster and achieved better validation scores when time constraints were shorter. For example, for the Abt-Buy dataset, including all-models takes 6000s to achieve 62% F1 Score but it only takes 1200s to achieve 62% F1 Score for AutoML-EM with random forest model. all-model ended up with higher validation scores because of the larger search space. This indicated that there were other better models for each dataset. For example,

Takeaways. *The model selection customization should be considered if the model is not running hours long, otherwise AutoML with full model space is preferable.*

3) AutoML-EM Pipeline Ablation Analysis: It is well recognized that hyperparameter tuning is an effective way to improve the model performance. Thus, in this experiment, we put our focus on the other two modules (data preprocessing and feature preprocessing).

To verify whether a module truly contributes to the final result, we did ablation experiments on the resulting pipeline of AutoML-EM. We selected the two most difficult datasets because there is a big performance gap between using and not using AutoML techniques. We followed the same setting as before: 3/5 for training, 1/5 for validation, and 1/5 for testing, and trained AutoML-EM for one hour to get a resulting pipeline. We reported the F1 score on the validation set after disabling data preprocessing and feature processing for the random forest model, respectively.

For example, consider an example pipeline generated by AutoML-EM in Figure 11. To disable its preprocessing module, we set `rescaling:__choice__`, `balancing:strategy` to `none`. Similarly, we can set `preprocessor:__choice__` to `no_preprocessing` to disable data preprocessing module.

Figure 12 shows the result. We can see that AutoML-EM with all modules performed the best on both datasets. Excluding data preprocessing and feature preprocessing modules leads to model performance degrading. For example, after excluding data preprocessing, the F1 Score dropped from 63.7 to 60.1 on the Amazon-Google dataset, and dropped from 63.9 to 56 on the Abt-Buy dataset. This result indicates that data preprocessing is an effective module for the two datasets. The F1 Score continued to drop after excluding the feature preprocessing module but not as dramatic as excluding data preprocessing. Please note that the purpose of this experiment is not to show that data preprocessing is more effective than feature preprocessing. We are sure that there will be cases where feature preprocessing plays a more important role.

Takeaways. *In the resulting pipeline of AutoML-EM, both data preprocessing and feature preprocessing modules contribute to the improved models.*

D. AutoML-EM-Active Evaluation

In this section, we aim to validate if AutoML-EM-Active is effective in the active learning setting for EM. Specifically, we compare the case where self-training is added as described in V-D2 and the case where only active learning is performed. We compare the two approaches under different parameter settings as described below. AutoML-EM-Active targets to improve model performance by labeling more data points with no extra human cost.

1) Parameters: **Initial training data size:** Initial training data is a random sample from the whole dataset. It is essential to have a good initial model for both active learning and self-training. We experiment on three values for initial data size, i.e., 30, 100, and 500, standing for small, medium, and large initial training data.

Dataset	Model Space	Validation Score on Time Constraint								Test Score on Time Constraint							
		60	300	600	1200	2400	3600	6000	8400	60	300	600	1200	2400	3600	6000	8400
BeerAdvo-RateBeer	all-model	93.6	95.9	97.2	97.9	100	100	100	100	80	80.3	76.5	80.8	72.8	79.3	77.5	78.8
	random forest	92.9	98.6	98.6	99.3	100	100	100	100	78.8	81.4	81.5	78.3	79.5	80.9	79.8	78.9
Fodors-Zagats	all-model	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	random forest	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
iTunes-Amazon	all-model	96.4	96	99.2	100	100	100	100	100	96	96.1	97.9	94.9	95.1	96.4	96.5	94.5
	random forest	97.2	98.4	99.2	100	100	100	100	100	95.3	95.5	94.9	96.9	96.5	95.7	96.4	97.8
DBLP-ACM	all-model	98.5	98.5	98.6	98.7	98.9	98.9	98.9	98.9	98.4	98.4	98.4	98.4	98.5	98.4	98.5	98.4
	random forest	98.7	98.7	98.7	98.7	98.9	98.8	99	99	98	98.2	97.9	98.1	98.3	98.1	98.2	97.9
DBLP-Scholar	all-model	94.5	94.5	94.6	95	95.1	95.2	95.4	95.5	94.2	94.2	94.4	94.8	95	95	95.3	95.2
	random forest	94.5	94.6	94.6	94.7	94.9	94.9	94.9	95.0	94.2	94.3	94.5	94.3	94.6	94.6	94.4	94.5
Amazon-Google	all-model	56.0	56.3	56.3	57.4	61.5	60.2	62.2	63.4	60.4	60.7	58.1	60.0	64.0	63.2	63.6	63.4
	random forest	56.1	57.1	59.7	61.7	62	61.8	62.6	63.1	60.1	61.5	61.9	65	64.9	63.8	64.7	64.6
Walmart-Amazon	all-model	78.2	78.2	78.8	79.2	82.1	83.5	83.6	83.9	78.2	78.2	78.6	77.2	79.1	78.6	79.8	80
	random forest	78.2	78.9	79.2	79.3	81.4	80.3	80.5	81.3	78.2	79.2	78.6	78.6	79	79.9	78.9	79.6
Abt-Buy	all-model	53.9	54.2	55.2	58	60.5	60.9	62.1	61.8	54.6	54.6	55.3	57.5	59	56.5	57.7	58.5
	random forest	55	59.1	60.8	62.2	62.8	62.6	63.5	63.2	55.9	56.1	56.9	58.9	58.8	58.1	58.7	57.8

Fig. 10: Model Selection for AutoML-EM. The darker the cell color, the higher the F1 Score.

```
Pipeline{
     'balancing:strategy': 'weighting',
     'rescaling:__choice__': 'robust_scaler',
     'rescaling:robust_scaler:q_min':
       0.19454891546620004,
     'rescaling:robust_scaler:q_max':
       0.9194022794180152,

     'preprocessor:__choice__':
       'select_percentile_classification',
     'preprocessor:select_percentile_classification:percentile':
       55.84285592896699,
     'preprocessor:select_percentile_classification:score_func':
       'f_classif',

     'classifier:__choice__': 'random_forest',
     'classifier:random_forest:bootstrap': 'True',
     'classifier:random_forest:criterion': 'gini',
     'classifier:random_forest:max_depth': 'None',
     'classifier:random_forest:max_features':
       0.9008519355763185,
     'classifier:random_forest:max_leaf_nodes': 'None',
     'classifier:random_forest:min_impurity_decrease': 0.0,
     'classifier:random_forest:min_samples_leaf': 2,
     'classifier:random_forest:min_samples_split': 6,
     'classifier:random_forest:min_weight_fraction_leaf': 0.0,
     'classifier:random_forest:n_estimators': 100,
   }
```

Fig. 11: Example AutoML-EM pipeline.

Dataset	AutoML-EM (Excluding DP and FP)	AutoML-EM (Excluding DP)	AutoML-EM
Amazon-Google	59.3	60.1	63.7
Abt-Buy	55.7	56.0	63.9

Fig. 12: AutoML-EM validation F1 Score by excluding modules (DP = Data Preprocessing, FP = Feature Preprocessing).

Dataset	Method	# of Active Learning Labels		
		40	160	400
Amazon-Google	AC + AutoML-EM	32.8	41.6	48.3
	AutoML-EM-Active	50.1	56.5	54.8
Abt-Buy	AC + AutoML-EM	34.0	39.7	45.2
	AutoML-EM-Active	42.8	45.1	52.9

Fig. 13: Comparing the test F1 Score between AutoML-EM and AC + AutoML-EM under different labeling budgets (init = 500 and st_batch = 200)

Active-learning batch size (ac_batch): Active learning batch size is the uncertain examples labeled at each iteration. This is the only human cost. We experiment on three values: 2, 8, and 20.

Self-training batch size (st_batch): We select a fixed batch of confident data examples for label inference. We experiment on four values: 0, 20, 50, and 200. Note that when st_batch is 0, it is equivalent to AC. We run both approaches for 20 iterations. A data scientist can customize the cut-off condition based on the labeling budget.

2) Evaluations: Recall that there are three types of datasets here: easy & small, easy & large, hard & large. For the easy datasets, a few iterations of active learning can already return a very good model. We show the evaluation on the two most difficult datasets (Amazon-Google and Abt-Buy). Our approach AutoML-EM-Active runs AutoML-EM on both active learning and self-training labels. We call the baseline approach AC + AutoML-EM, which is to run AutoML-EM on the active learning labels only.

We first examine how effective self-training is to AutoML-EM. We varied the number of active learning labels and compared the test F1 Score between AutoML-EM-Active and AC + AutoML-EM (init = 500 and st_batch = 200). Figure 13 shows the comparison results. We can see that AutoML-EM-Active significantly outperformed AC + AutoML-EM. For instance,

Dataset	Method	init = 30	Init = 100	init = 500
Amazon-Google	AC + AutoML-EM	47.6	48.1	48.3
	AutoML-EM-Active	32.3	53.5	54.8
Abt-Buy	AC + AutoML-EM	48.2	43.2	45.2
	AutoML-EM-Active	45.2	53.1	52.9

Fig. 14: Comparing the test F1 Score between AutoML-EM and AC + AutoML-EM under different initial training data size (ac_batch = 20 and st_batch = 200).

Dataset	AC + AutoML-EM	AutoML-EM-Active		
		st_batch = 20	st_batch = 50	st_batch = 200
Amazon-Google	48.3	48.7	53.6	54.8
Abt-Buy	45.2	45.2	46.8	52.9

Fig. 15: Comparing the test F1 Score between AutoML-EM and AC + AutoML-EM under the self-training batch size (st_batch) (init = 500 and ac_batch = 2).

when the number of active learning labels is 160, AutoML-EM-Active had an F1 Score of 56.5 while AC + AutoML-EM's F1 Score is 41.6. This result validated the effectiveness of self-training.

We next examine the impact of the initial training size (init) on AutoML-EM-Active's performance. We varied init and compared the test F1 Score between AutoML-EM-Active and AC + AutoML-EM. The results are shown in Figure 14. We can see that when the initial training data size is large (i.e., init = 100 and 500), the label inference accuracy is high, so that AutoML-EM-Active helped to improve the active learning process effectively. When init = 30, the initial model is of very low quality. In this situation, self-training should not be applied since it will infer many wrong labels.

Finally, we explore how the number of inferred labels affect the results. We evaluated the test F1 Score of AutoML-EM-Active by varying the self-training batch size (st_batch). Figure 15 shows the result. Note that when st_batch = 0, since there is no labels inferred in this situation, AutoML-EM-Active (st_batch = 0) is equivalent to AC + AutoML-EM. We can see that as st_batch increased, AutoML-EM-Active's performance got improved, but the improvement could get less and less. For example, on the Amazon-Google dataset, F1 Score was increased by 53.6% - 48.7% = 4.9% when st_batch was increased from 20 to 50 but it only increased by 54.8% - 53.6% = 1.2% after increasing from st_batch = 50 to 200.

Takeaways. *Self-training effectively improves AutoML-EM in the active learning setting if the initial training size is not very small (e.g., init* > 100*).*

VI. Related work

ML model development for EM: In addition to Magellan [31] and DeepMatcher [28], there are some other recent efforts to study how to develop an ML model for EM. For example, DeepER [14] proposed an end-to-end deep learning solution for EM. As shown in [28], it is subsumed by the deep learning design space used by DeepMatcher. Auto-EM [41] leverages transfer learning to reduce data labeling cost, which is orthogonal to our work. To the best of our knowledge, we the first to study how to automatically build an ML model for EM using AutoML.

Active learning for EM: There is a long history [5], [26], [32], [35] of applying active learning to the EM problem to reduce the human labeling cost. The focus is to get a good model with as few labels as possible. While we also use active learning, our focus is not to invent a better query strategy for active learning, but to explore how promising to combine active learning with self-training for EM.

Semi-supervised learning for EM: Semi-supervised learning falls between supervised learning and unsupervised learning, targeting at the scenario where only a small amount of labeled data and a large amount of unlabeled data are available. It enlarges the labeled dataset given the unlabeled dataset. There are many algorithms have been developed in the ML community [42]. To the best of our knowledge, there is only one paper [21] that studied applying self-training for EM. But we have seen that self-training alone is hard to get a competitive model for hard EM datasets. Our point here is to strengthen active learning with self-training. We leave studying other label inference approaches as future work.

Data labeling with weak supervision: Weak supervision approaches proposed to adopt labeling functions (specified by the user [33] or iteratively learned from data [38]) to construct a noisy training dataset, and then train a noise-tolerant model. Unlike these works, we study how to combine self-training and active learning for EM.

VII. Conclusion and future work

We studied how to automate entity matching model development with AutoML. We justified why AutoML is needed for EM, and proposed AutoML-EM, a customized AutoML solution for EM. We studied how to integrate AutoML-EM into the active learning setting and proposed AutoML-EM-Active, a hybrid framework to combine active learning and self-training for AutoML-EM. The results showed that i) AutoML-EM outperformed human-developed models by a large margin; ii) AutoML-EM reached or exceeded deep learning models even on textual data; iii) AutoML-EM-Active is a more effective framework than AC + AutoML-EM when the initial training size is not very small.

Despite the results, AutoML-EM has limitations. For example, DeepMatcher is better at handling long text data inherently due to the benefits of adopting NLP techniques. A concurrent work [24] brings the state-of-the-art using the pre-trained deep learning model. There are many interesting future research directions to be further explored. First, AutoML-EM may produce a model that is hard to explain. We would like to explore how to leverage recent ML explanation tools (e.g., Shap [25] and Lime [34]) to help data scientists to understand a complex EM model. Second, AutoML-EM could take a long time to find the very best model in the large search space. Meta-learning [37], which learns how to design a model from

historical ML tasks, is a promising idea. We plan to investigate whether it can be applied to speed up AutoML-EM. Third, AutoML-EM-Active is a general idea to combine active learning and self-training for AutoML-EM. This paper shows the great potential of this idea. We would like to extend it to other active learning algorithms, such as query by committee and maximum margin, in the future.

VIII. Acknowledgment

This work was supported in part by Mitacs through an Accelerate Grant, NSERC through discovery grants and a CRD grant. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1]
[2] Dedupe.io. https://dedupe.io/.
[3] sklearn.preprocessing.RobustScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html.
[4] TPOT. https://epistasislab.github.io/tpot/.
[5] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 783–794, 2010.
[6] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1131–1139, 2012.
[7] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.
[8] E. Bringer, A. Israeli, A. Ratner, and C. Ré. Osprey: Weak supervision of imbalanced extraction problems without code. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, page 4. ACM, 2019.
[9] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1335–1349, 2020.
[10] X. Chen, Y. Xu, D. Broneske, G. C. Durand, R. Zoun, and G. Saake. Heterogeneous committee-based active learning for entity resolution (healer). In *European Conference on Advances in Databases and Information Systems*, pages 69–85. Springer, 2019.
[11] P. Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 151–159. ACM, 2008.
[12] P. Christen. The data matching process. In *Data Matching*, pages 23–35. Springer, 2012.
[13] X. L. Dong and T. Rekatsinas. Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1645–1650. ACM, 2018.
[14] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.
[15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.
[16] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.
[17] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.
[18] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 601–612. ACM, 2014.
[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.
[20] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning*. Springer, 2019.
[21] M. Kejriwal and D. P. Miranker. Semi-supervised instance matching using boosted classifiers. In *European semantic web conference*, pages 388–402. Springer, 2015.
[22] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.
[23] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.
[24] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.
[25] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.
[26] V. V. Meduri, L. Popa, P. Sen, and M. Sarwat. A comprehensive benchmark framework for active learning methods in entity matching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1133–1147, 2020.
[27] B. Mozafari, P. Sarkar, M. Franklin, M. Jordan, and S. Madden. Scaling up crowd-sourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment*, 8(2):125–136, 2014.
[28] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34. ACM, 2018.
[29] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016.
[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[31] S. D. Pradap Konda, P. S. GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, et al. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12), 2016.
[32] K. Qian, L. Popa, and P. Sen. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1379–1388, 2017.
[33] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.
[34] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
[35] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, 2002.
[36] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.
[37] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.
[38] P. Varma and C. Ré. Snuba: automating weak supervision to label training data. *Proceedings of the VLDB Endowment*, 12(3):223–236, 2018.
[39] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 229–240, 2013.
[40] Q. Yao, M. Wang, Y. Chen, W. Dai, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.
[41] C. Zhao and Y. He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*, pages 2413–2424. ACM, 2019.
[42] X. J. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.
[43] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.