Minun: Evaluating Counterfactual Explanations for Entity Matching

Jin Wang
jin@megagon.ai
Megagon Labs
United States

Yuliang Li
yuliang@megagon.ai
Megagon Labs
United States

ABSTRACT

Entity Matching (EM) is an important problem in data integration and cleaning. More recently, deep learning techniques, especially pre-trained language models, have been integrated into EM applications and achieved promising results. Unfortunately, the significant performance gain comes with the loss of explainability and transparency, deterring EM from the requirement of responsible data management. To address this issue, recent studies extended explainable AI techniques to explain black-box EM models. However, these solutions have the major drawbacks that (i) their explanations do not capture the unique semantics characteristics of the EM problem; and (ii) they fail to provide an objective method to quantitatively evaluate the provided explanations. In this paper, we propose Minun, a model-agnostic method to generate explanations for EM solutions. We utilize counterfactual examples generated from an EM customized search space as the explanations and develop two search algorithms to efficiently find such results. We also come up with a novel evaluation framework based on a student-teacher paradigm. The framework enables the evaluation of explanations of diverse formats by capturing the performance gain of a "student" model at simulating the target "teacher" model when explanations are given as side input. We conduct an extensive set of experiments on explaining state-of-the-art deep EM models on popular EM benchmark datasets. The results demonstrate that Minun significantly outperforms popular explainable AI methods such as LIME and SHAP on both explanation quality and scalability.

ACM Reference Format:
Jin Wang and Yuliang Li. 2022. Minun: Evaluating Counterfactual Explanations for Entity Matching. In *Proceedings of Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning, June 12–17, 2022, Philadelphia, PA (DEEM@SIGMOD '22).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Entity matching (EM), a.k.a entity resolution, record linkage, reference reconciliation, and duplicate detection, refers to the problem of identifying pairs of data entries that represent the same real-world entity [18, 20]. It is a fundamental problem of data integration and has a broad scope of real-world applications. Recently deep learning techniques have been widely adopted in data integration tasks including EM [36] due to their superior performance and the ability to avoid intensive feature engineering efforts. Among them, EM solutions that utilize pre-trained Transformer models (LM) [4, 19, 27] have achieved state-of-the-art results.

Nevertheless, the performance of deep learning based approaches is at the cost of reducing transparency and interpretability. While classic EM methods relying on traditional machine learning or string similarity measurement are naturally interpretable, deep learning based approaches are more like black-box models that are barely interpretable nor provide sufficient explanations of the outcome. In real application scenarios, however, it is essential to ensure that human users can interpret these models and understand why they make certain positive/negative predictions. For example, businesses can benefit from the models being more transparent to trust them in decision making; while the end-users can gain more insights and confidence in the models' predictions. More recently, the data management community has also recognized the necessity of responsible data management [34]. Following this trend, there is a series of studies trying to provide explainable results for data management applications. Based on whether the internal structure of the target model is available when making the explanations, the explanation methods can be categorized into the *black-box* and *white-box* based methods [14]. White-box methods exploit the structural properties of the models, such as decision trees, to generate the explanations; while for black-box methods, the target model is only available as an oracle that returns a prediction of the given input. Besides, the *global explanation* methods aim at systematically explaining the behavior of a model, while the *local explanation* methods focus on explaining the model's behavior for each given instance [14].

To satisfy the requirement of responsible data management, many efforts have been made to explain the results of deep learning based EM approaches [2, 3, 5, 8]. The main methodology of them is to extend black-box explanation methods from the field of explainable AI to provide local explanations, i.e. given the prediction of an input pair of entities, explain why they match or do not match. The explanation typically comes in the format of which elements from the input contribute most to the matching decision. However, these methods suffer from two major flaws: First, they failed to identify a proper granularity of the explanations. Generally speaking, they provide explanations at two levels: the token level and the attribute level. However, in practice, these two types of explanations alone are not expressive enough to capture the rich matching-based semantics of EM tasks. Instead, it is usually necessary to have a finer-grained explanation type that captures the interaction between key tokens or attributes. Selecting the right level of granularity is also key to the performance of explanation methods. Second, they did not provide a reasonable way to quantitatively evaluate the quality of explanation results. Most of them relied on case studies to illustrate the explanations; while the quantitative studies made by some methods [2, 3] are specific to their own explanation framework and cannot be easily extended to other ones. As a result, it is hard to compare the quality of explanations from different explanation approaches.

In this paper, we propose Minun, a model-agnostic framework to provide local explanations for black-box EM models. Specifically, we utilize counterfactual examples of entity pairs as the explanation. Following previous studies, a counterfactual example in EM application is a new instance that can flip the prediction by the black-box model on the original input instance. There are some existing studies in the data management field that aim at finding counterfactual explanations. Two typical examples are Geco [32] and Lewis [12]. However, they cannot be directly applied to EM applications because (i) currently they only support tabular data with numerical and categorical attributes, while EM requires to work on one or several textual attributes; (ii) they are designed for single-record classification while EM is a matching task requiring classification of entity or sequence pairs. As a result, it is rather difficult for them to capture the pair-wise interactions between the input pairs of entities.

To address such challenges, we regard tokens in two entity entries as the basic unit of explanations and develop a token-level edit distance metric to describe the search space of the problem. On the basis of it, we further define explanations to be sequences of operations on one entity entry that make it more similar to the other entry to flip the negative prediction made by the black-box model to positive (or vice versa). In this case, the explanations generated by our framework become pairwise and can capture semantically richer matching signals between the two entity entries. Under this framework, we first propose a simple greedy search algorithm that enumerates the edit operations on each pair of attributes respectively. To accelerate this process, we further develop a more efficient binary search algorithm to skip any unnecessary candidates.

Moreover, we also develop a new evaluation framework to judge the quality of explanations based on a recent trend developed in NLP [15, 29]. The basic idea is that we utilize the generated explanations to construct a training set to train a new model different from the black-box model to be explained. Then the black box model can be regarded as a "teacher" that teaches the "student" model using knowledge from the generated explanations. Consequently, the quality of a set of explanations can be objectively measured by the performance of the student model. We further show that the proposed framework is a general solution as it can capture a wide variety of explanation methods including LIME [30], SHAP [21], and various counterfactual explanation methods.

In summary, this paper makes the following contributes:

- We propose Minun, a model-agnostic framework to explain the results of entity matching applications with the help of counterfactual examples.
- We present a novel method to describe the counterfactual explanations for sequence pairs as well as textual records. We also propose two efficient search algorithms for those explanations.
- We develop an evaluation framework that allows researchers to quantitatively measure the quality of explanations for EM applications.
- We evaluate our proposed methods on several popular EM datasets using the evaluation framework. The results demonstrate that Minun generates higher-quality counterfactual explanations while being more scalable versus popular explainable AI solutions such as LIME or SHAP.
- We will open-source Minun at https://github.com/megagonlabs/minun.

The rest of this paper is organized as follows: Section 2 introduced the background and formally defines the problem. Section 3 proposed two algorithms to efficiently find the counterfactual explanations. We present in Section 4 a novel evaluation framework to judge the quality of the generated explanations. Section 5 shows the experimental results. We survey related work in Section 6 and conclude in Section 7.

2 PRELIMINARY

2.1 Terminology

We first introduce the necessary terminologies to formally describe the problem.

Given a pair of entity entries, the entity matching (EM) problem aims at identifying whether the two entity entries refer to the same real-world object. We denote the pair record as $p = \langle E_L, E_R \rangle$ that consists of two entity entries (left and right) and the label l for them can be 1 (match) or 0 (unmatch). Here an *entity entry* E consists of a set of attributes where each *attribute* a is a sequence of tokens. An entity entry E with n attributes can be denoted as $E = \{a_1, a_2, \ldots, a_n\}$. The i-th attribute in the left/right entity is denoted as a_i^L and a_i^R respectively. Correspondingly, an attribute a_i with m tokens can be denoted as $a_i = [t_1, t_2, \ldots, t_m]$. The length of an attribute is defined as the number of tokens in it. In this paper, we assume that the two entity entries in a pair are structural and with the same or pre-aligned attributes, following the settings of previous studies [18, 19, 24].

For learning-based EM, the goal is to train a binary classifier M such that for every pair $p = \langle E_L, E_R \rangle$, $M(p) = 1$ if E_L matches E_R and 0 otherwise. For the standard supervised learning setting, the classifier M is first trained on a training dataset D_{Train} of labeled entry pairs then evaluated on an unseen test dataset D_{Test} by computing the F1 scores. For explanations in this paper, we assume the settings of *black-box* explanations, meaning that the only interaction between an explanation method (i.e., the explainer) with the target EM model M is by querying M as an oracle with pairs of entity entries to obtain the binary predictions.

2.2 Counterfactual Explanation

Suppose we have a black-box model for EM denoted as M, an oracle that can make a binary prediction 0/1 when given a pair of entity entries. Given an entity pair p, a counterfactual example of p denoted as p', which is generated by applying certain perturbations over p, can flip the prediction of the black-box model M, i.e. $M(p) \neq M(p')$. In a nutshell, counterfactual explanations

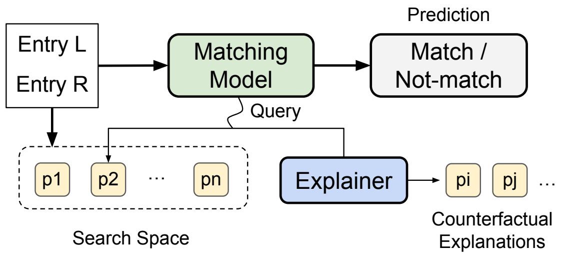


Figure 1: An overall architecture of counterfactual explanation search algorithms.

describe the *minimal changes to change the model's prediction*. To formally define the task, one needs to identify the *search space* and *distance metrics*, as in previous studies [21, 30, 31]. The search space describes the valid (sequence of) perturbations to obtain p' from p, while a properly chosen distance metrics ensures that p and p' are naturally close together according to the application's needs. We have the following generic problem definition:

Definition 2.1. Given a black-box model M, an input instance p, a search space Φ, and a distance metrics D, the goal of counterfactual explanation is to compute counterfactual example $p' \in \Phi(p)$ such that $M(p') \neq M(p)$ and p' minimizes $D(p, p')$.

2.3 Baseline Approaches

Next, we describe baseline approaches for generating counterfactual explanations for EM. Given the search space Φ, the black-box model M, and an input instance p, a generic baseline solution is to traverse the search space $\Phi(p)$ in ascending order of the distance metrics and evaluate for each $p' \in \Phi(p)$, whether the prediction can be flipped, i.e., $M(p) \neq M(p')$. Note that applications may require a set of counterfactual examples as output. So the loop continues until it finds enough results or all candidates have been checked. Figure 1 shows the overall architecture of this generic process.

Without further optimizations, we can come up with several baseline solutions for EM based on the above template. First, we define the search space based on token-level or span-level transformations. Here we treat each record p as a token sequences that consist of the tokens from the two entity entries. Motivated by previous studies [30, 31] that remove tokens to create candidate explanations, the first baseline method generates candidates by removing spans of tokens with a fixed length q from the token sequence. The search space can be constructed by using a sliding window of length q over an entity entry to enumerate the spans to be removed.

One issue to be resolved in the first baseline is that the removal operations typically make the two entity entries more dissimilar. This means that the generated explanations can only flip the positive predictions by the black-box model (we will call them "positive pairs" and the ones with negative predictions "negative pairs" next). For negative pairs, we need a separate search space that generates candidates that make the two entity entries more similar. To this end, we propose the second baseline as follows. For negative pairs of entity entries, we generate the candidates by copying spans of tokens with a fixed length q from one entity to the other one. A combined version of the baseline can generate candidates from the union of the two search spaces by applying the removal operation on the positive pairs and the copy operation for the negative pairs.

2.4 Extending the search space

The above baseline methods are limited as they can only generate simple explanations. Indeed, in our experiment, explanations based on 3-gram have low coverage of only 8.2% of the instances in the datasets, i.e., >90% of all the instances cannot be explained by token-level operations of 3-grams. To address this challenge, our first step is to extend the search space by providing a richer set of transformations to improve the explanation coverage for EM.

The core idea is the following. Instead of treating the input p as a single sequence as in previous studies of explaining black-box NLP models, we describe the explanations for a given pair p as transformations from the left entity entry to the right (or vice versa). If $M(p)$ is 1, then the transformations should make the left entry less similar to the right entry so as to flip the prediction to 0; similarly, if $M(p)$ is 0, the transformations should make the left entry more similar with the right one. Each unit transformation can be defined as a token-level operator, e.g., token removal, insertion, or replacement. To support the unique characteristics of EM as a matching task, we also allow transformations to be attribute-level operators such as copying tokens from the right entry to the left highlighting the difference between the two attributes. Under this definition of transformations, we can restrict the search space to be within a tractable scale. In addition, we can naturally define the distance metrics to be the edit distance, namely the number of transformation steps needed to reach p' from p. We formally define our search space and search algorithms in Section 3.

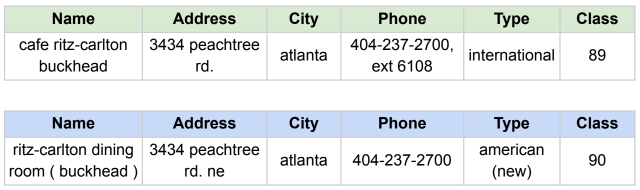
The motivation of making the above search space definition is two-fold: first, previous studies of counterfactual explanations [12, 32] only support categorical and numerical features. As such, their problems of finding counterfactual examples naturally has a finite search space. However, in EM applications, most features are textual which can result in a search space of infinite size for the candidate p'. By introducing the attribute-wise transformation, we define a finite search space with reasonable semantics for textual features in EM applications. Second, since EM is a pairwise matching task, i.e., it can be formulated as classification of sequence pairs, the explanations should capture similarity and dissimilarity between the two input entries. Our extended search space definition provides exactly such transformation as we highlighted above.

3 METHODOLOGY

In this section, we introduce the algorithms to find counterfactual explanations for EM applications. We first formally define the search problem in Section 3.1. Then we propose a greedy algorithm in Section 3.2 and a binary search optimization in Section 3.3.

3.1 Problem Definition

To formulate the problem of explaining EM, we first need to define the search space. Based on the above discussion, the search space is decided by the transformations between attributes of the left and right entities. To this end, we introduce the definition of *token-level edit distance*, which is a generalization of the character-level edit distance widely used in measuring string similarity [13]. Similar to character-level edit distance, there are three basic operations in token-level edit distance: insertion, deletion, and substitution. The distance definition is the minimum number of operations to transform one token sequence into another. The only difference is that the operations are performed on tokens instead of characters. The computation of token-level edit distance can be done using dynamic programming within $O(n^2)$ time, where n is the number of tokens of the two input sequences. In the rest of this paper, we will use edit distance for short if there is no ambiguity. Given two token sequences S and T, we denote their edit distance as $\mathsf{ED}(S, T)$.



Name	Address	City	Phone	Type	Class
cafe ritz-carlton buckhead	3434 peachtree rd.	atlanta	404-237-2700, ext 6108	international	89

Name	Address	City	Phone	Type	Class
ritz-carlton dining room (buckhead)	3434 peachtree rd. ne	atlanta	404-237-2700	american (new)	90

Figure 2: An example of entity pairs labeled as “unmatch”.

Next, to explain black-box EM models, we apply the transformation operators defined above on each pair of aligned attributes of the input entries. Note that the edit distance between the left and right entries can be calculated by summing up those between each pair of aligned attributes. As edit distance is symmetric, without loss of generality, we assume that all operations will be performed to transform the left entity to the right one. For each attribute a_i, the candidate explanations are generated by applying 0 to $\mathsf{ED}(a_i^L, a_i^R)$ edit operations. The search space can be formulated as the combination of all possible transformations from all attributes. Given an entity pair p, we denote the search space constructed correspondingly as $\Phi(p)$. Following the idea of previous studies of finding counterfactual explanation [32], a good explanation should have fewer different features from the original instance. Therefore, in our search problem definition, we also aim at finding the candidates that can flip the predictions while having the smallest edit distance from the original input.

Example 3.1. Given a pair of entities shown in Figure 2, the edit distance on the attribute a_2 *address* is 1, where the right entity can be transformed from the left one by inserting the token “ne” after the token “rd.”. The edit distance between each pair of attributes is 5,1,0,0,4,1, respectively. The edit distance between the two entries is $5 + 1 + 0 + 0 + 4 + 1 = 11$ and there are $6 \times 2 \times 1 \times 1 \times 5 \times 2 = 120$ possible candidates in the search space.

Based on the above discussion, we can formally define the search problem as follows.

Definition 3.2. Given a pair of entity $p = \langle E_L, E_R \rangle$ and a black-box EM model M, the problem of explaining EM aims at finding a set of explanations $P \subseteq \Phi(p)$ satisfying that $\forall p' \in P$, $M(p) \neq M(p')$ and for any candidate $p'' \in \Phi(p) - P$ and $p' \in P$, $\mathsf{ED}(p', p) \leq \mathsf{ED}(p'', p)$.

Note that the above definition captures finding a set of (more than 1) counterfactual explanations. The number of explanations, i.e. the carnality of P in above definition is a hyper-parameter and can be set by the user based on the real application scenario.

3.2 Greedy Algorithm

Next, we introduce the search algorithms for finding counterfactual explanations using the search space and distance metrics defined above. For negative pairs, we will use all three edit operations: insertion, deletion and substitution; for positive pairs, as the set of transformations should make the two entries more dissimilar, we only use the deletion operation, i.e. delete a token at one time. Suppose each entity entry has n attributes, then a state in the search space is an n-dimensional vector where the value of the i-th dimension is the number of edit operations applied on the corresponding attribute. To reduce the potential computation overhead, we just apply the edit operations on each attribute sequentially and do not enumerate the combination of them. There are potential optimization opportunities in this step, e.g., assigning weights to tokens or using external knowledge to decide the order to search the attributes. Due to the space limitation, we will just describe the default approach and leave further optimizations as future work.

Algorithm 1: Greedy(p, M, K)

```
Input: $p$: The pair of entity entries; $M$: The black box model;
       $K$: The number of explanations to be collected
Output: $P$: The set of explanations
1  Initialize $P \leftarrow \emptyset$;
2  for $i = 1$ to $n$ do
3      Compute the edit distance $\mathsf{ED}(a_i^L, a_i^R)$;
4  Formulate $\Phi(p)$ based on $\{\mathsf{ED}(a_i^L, a_i^R)\}_{i \leq n}$;
5  Sort $\Phi(p)$ in the ascending order by the number of edit
     operations;
6  foreach state in $\Phi(p)$ do
7      Generate the candidate $p'$ from the state;
8      if $M(p) \neq M(p')$ then
9          $P \leftarrow P \cup \{p'\}$;
10     if $|P| == K$ then
11         return $P$;
12 return $P$;
```

Following this route, we then propose a greedy algorithm to traverse the search space in Algorithm 1. We first initialize the result set P as an empty set (line 1). Then we calculate the edit distance between each pair of attributes in the left and right entries of p, respectively (line 3). As mentioned before, if $M(p)$ is 1, we will only use deletion to make two attributes more dissimilar and the computation cost is $O(n)$ then. After obtaining the edit distance between each pair of attributes, we can then construct the search space by enumerating the state vectors. Since we want to find the counterfactual explanations that are most similar with the original instance, we sort the state vectors in ascending order by the total number of operations applied in all dimensions of the state vector (line 5). If there is a tie, we prioritize the state vectors with fewer attributes. Then we generate the candidate from each state vector and evaluate it with the black-box model. If the prediction is flipped, then we add the candidate into the result set (line 9). If we have collected K results (line 11) or finish traversing the search space (line 12), the search process will end.

Example 3.3. We look at the pair in Figure 2 again. We will start with the state vectors with 1 edit operation in total, i.e. $\langle 1,0,0,0,0,0\rangle$. To generate the candidate from it, we need to apply one edit operation on attribute "name", where it should be the deletion on token "cafe". Then we get the candidate as *ritz-carlton buckhead, 3434 peachtree rd., atlanta, 404-237-2700, ext 6108 international, 89* (different attributes are separated with comma.) and will evaluate it using the black-box model M to decide whether it can flip the prediction on p. Then we move to the next state vector $\langle 0,1,0,0,0,0\rangle$ with 1 operation in total, which generates the candidate by inserting the token "ne" in the end of the "address" attribute.

3.3 Optimization via Binary Search

Although the greedy algorithm can correctly return the counterfactual explanations with the fewest edit operations, it requires enumeration of the search space resulting in a large computational overhead. This cost can be reduced by approximations. Our optimization is based on the observation that during the search process, the target model M is *approximately monotonic* along each dimension of the state vectors. This means that for each attribute a_i, the more edit operations we apply, the more likely that the prediction $M(p')$ will be "pushed" to the right direction, i.e., $M(p') \rightarrow 1$ if $M(p) = 0$ or vice versa. This is based on the assumption that the model M is "reasonable", meaning that increasing the similarity between the left and right values of an attributes a_i will only increase the chance of M predicts positive, and vice versa.

Under this assumption, when looking at the d-th dimension of a state vector with values on the other dimensions fixed, if the prediction $M(p')$ with x edit operations cannot flip the original prediction, then applying 0 to $x-1$ operations will not flip the prediction either. In other words, it could help accelerate the search process if such states can be skipped. Based on this observation, we come up with the following binary search algorithm.

The binary search based approach is shown in Algorithm 2. The initial first steps are similar to those in the greedy algorithm. The difference lies in that we use binary search to traverse the search space. We denote by Search(A, lb, ub, num) a function call to the binary search procedure. The input arguments consist of:

- A: the set of attributes to be searched on.
- lb and ub: the set of lower bounds and upper bounds of all involved attributes to be searched.
- num: the number of involved attributes.

The Search function works as follows: it starts from the state vector where the value of dimension d equals $\frac{1}{2}(d.lb + d.ub)$ and generates the corresponding candidate p'. If p' can flip the prediction (i.e., $M(p') \neq M(p)$), we will add the candidate into the results and decrease the upper bound. Otherwise, it means that more edit operations need to be applied so we will increase the lower bound in a binary search manner. The same process keeps iterating on each attribute until there is no more candidate left in between the lower and upper bounds, i.e. until the lower bound is no smaller than the upper bound on all attributes in A.

Here we enumerate the state vectors in the ascending order of the number of attributes involved in the computation. When only one attribute is involved, we directly perform binary search on that attribute (line 7). If the number is larger than 1, we need to enumerate all possible combinations of attributes (line 9). For each combination G, we first initialize the lower and upper bounds on each dimension (line 13). Then we start from the middle point where the value on each dimension is half of the edit distance between the two attributes and iteratively perform binary search on each involved attribute until the Search function terminates (line 14). The overall stopping condition is the same as that of Algorithm 1.

```
Algorithm 2: Binary Search(p, M, K)
   Input: p: The pair of entity entries; M: The black box model;
          K: The number of explanations to be collected
   Output: P: The set of explanations
 1 Initialize P ← ∅;
 2 for i = 1 to n do
 3     Calculate the value of $\mathsf{ED}(a_i^L, a_i^R)$;
 4 for j = 1 to n do
 5     if j = 1 then
 6         for i = 1 to n do
 7             $P \leftarrow P \cup \text{Search}(\{a_i\}, \vec{0}, \{\mathsf{ED}(a_i^L, a_i^R)\}, 1)$;
 8     else
 9         $\mathcal{G}$ ← Enumerate all combinations with j attributes;
10         foreach $G \in \mathcal{G}$ do
11             $B.lb \leftarrow \vec{0}$;
12             foreach $a_i \in G$ do
13                 $B.ub[i] \leftarrow \mathsf{ED}(a_i^L, a_i^R)$;
14             $P \leftarrow P \cup \text{Search}(G, B.lb, B.ub, j)$;
15     if |P| ≥ K then
16         return P;
17 return P;
```

Note that the above binary search based approach cannot guarantee to return the exact results with the fewest edit operations. This is because the algorithm enumerates the candidates in ascending order of the number of involved attributes instead of the number of operations. Nevertheless, we will show later in the experiments that explanations discovered by this approach have comparable or even better quality compared to the ones returned by the greedy approach. In fact, we notice that the new enumeration strategy can provide a more diverse set of explanations. We expect that this property can benefit certain applications. Thus, the binary search version is also a reasonable alternative to the exact greedy algorithm in terms of explanation quality.

Example 3.4. Suppose we are at the point of checking the combination of attributes "name", "type" and "class". Since the edit distance on these attributes are 5, 4 and 1, respectively, the total number of state vectors to be checked is $6 \times 5 \times 2 = 120$. By applying the binary search strategy, we start from the state vector $\langle 2,0,0,0,2,0\rangle$, with the lower bound of each attribute as 0 and upper bound as 5,4,1, respectively. If the corresponding candidate cannot flip the prediction, we will change the lower bound of each attribute one by one: first we set the lower bound of "name" to 3 and move on to the state vector $\langle 4,0,0,0,2,0\rangle$. Suppose now that the corresponding candidate can flip the prediction, we stop at searching this attribute and add candidates corresponding to both $\langle 4, 0, 0, 0, 2, 0\rangle$ and $\langle 5, 0, 0, 0, 2, 0\rangle$ into the result set. Next, we will set the lower bound of "type" to 3 and start from the state vector $\langle 2, 0, 0, 0, 3, 0\rangle$ in a similar way. Since this process is performed recursively, there is also a search path starting from $\langle 4, 0, 0, 0, 2, 0\rangle$ by varying the values in the last two attributes. In this way, the number of state vectors to be checked will be much smaller than 120, the carnality of search space.

4 EVALUATION FRAMEWORK

4.1 Methodology

A key challenge in Explainable AI (XAI) is the evaluation of explanations. While many popular model explanation tools such as LIME [30] and SHAP [21] have been developed and widely applied, the community has not yet reached a consensus on how to rigorously evaluate different explanation methods. Some general guidelines [7] (also see Chapter 3.5 of [23]) exist such as evaluating expressiveness, fidelity, comprehensibility, etc. of the explanations. However, it has not been formalized how each aspect can be quantitatively calculated.

In the context of EM applications, the evaluation task only gets more challenging because of the *diverse format* of explanations. Classic methods like LIME and SHAP output explanations in the format of feature importance, i.e., they associate a weight to each token in the entity entries. These weights can further be aggregated into attribute importance such as in [5]. Counterfactual explanations, as we describe in Section 3, come in the format of modified counterfactual examples from a customizable search space. Unlike some traditional tasks from NLP and CV like text/image classification, there is not a well-defined format of explanations for EM applications. As a result, it is not feasible to create ground truth explanations via human annotation.

In Minun, we design an evaluation framework for Explainable EM solutions (i.e., explainers) following a recent trend developed in the NLP field [15, 29]. To quantitatively evaluate the quality of explanations, Hase and Bansal [15] proposed to measure by how much these explanations can *help "students" to simulate the target model's behavior (i.e., the "teacher") on unseen examples.* The "students" can be human subjects drawn from the target users' population as in [15], while Pruthi et al. [29] proposed to replace human subjects with a machine learning model to avoid human bias and to enable automatic evaluation.

In a nutshell, the experiment consists of two rounds of simulation processes. Both rounds simulate a scenario where the student model learns from unseen examples labeled by the teacher model. The only difference between the two rounds is that the second round *provides the explanations as side information* to the student in addition to its training data while the first round does not. Intuitively, high-quality explanations by definition should provide more insights to users to help better understand the target model's behavior. Therefore, we can objectively quantify the quality of the explanations by the performance gain (i.e., improvement of F1 scores for EM) when the explanations are present versus when they are not. Figure 3 shows a high-level view of the experiment design.

We formally describe the evaluation process in Algorithm 3. The process starts with taking an unlabeled dataset D and annotating it with match/unmatch labels using the target (teacher) model T (Line 1). The dataset is further split into training and test sets for the student models S and S'. Note that D needs to be disjoint from the training set for the target T to avoid any label leakage via the explanations. The first round is a standard training loop of an EM classifier (Line 3-8). The student S is updated via back-propagation for a fixed number of epochs (or until convergence).

The second round (Line 9-15) is the same training loop except that in Line 13, it uses the explainer E (e.g., LIME, SHAP, Minun) to augment each example of the training batch B. This is done by a special data augmentation operator aug that augments the batch with knowledge in the explanations $E(B)$.

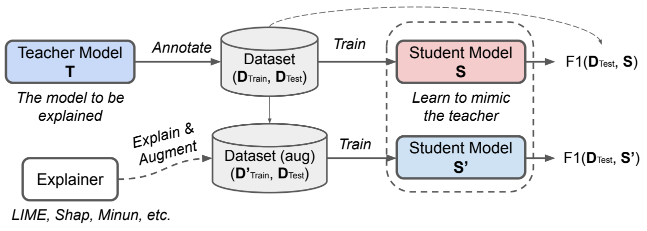


Figure 3: Evaluation Framework

Algorithm 3: Teacher-Student Simulation Process.

```
Input: A target teacher EM model $T$;
  An unseen, unlabeled EM dataset $D$;
  An Explainer $E$ (e.g., LIME, Minun); A transformation operator aug
Variables : A student model $S$;
  Number of training epochs n_epoch;
  Learning rate $\eta$
Output: $\Delta F_1$, the measured explanation quality
/* Annotate and split $D$ into training and test sets */
1  $D \leftarrow$ annotate$(D, T)$;
2  $D_{\text{Train}}, D_{\text{Test}} \leftarrow$ Split(D) ;
/* Round 1: no explanation */
3  Initialize the student $S$;
4  for ep = 1 to n_epoch do
5      Randomly split $D_{\text{Train}}$ into mini-batches $\{B_1, \dots B_n\}$;
6      for $B \in \{B_1, \dots B_n\}$ do
           /* Back-prop to update $S$ */
7          $\mathcal{L} \leftarrow$ CrossEntropy$(S, B)$;
8          $S \leftarrow$ back-propagate$(S, \eta, \partial\mathcal{L}/\partial S)$;
/* Round 2: with explanation */
9  Re-initialize the student $S'$;
10 for ep = 1 to n_epoch do
11     Randomly split $D_{\text{Train}}$ into mini-batches $\{B_1, \dots B_n\}$;
12     for $B \in \{B_1, \dots B_n\}$ do
           /* Augment the batch $B$ using the explanations
              (see Table 1) */
13         $B_{\text{aug}} \leftarrow B \cup$ aug$(B, E(B))$;
           /* Back-prop to update $S'$ */
14         $\mathcal{L} \leftarrow$ CrossEntropy$(S', B_{\text{aug}})$;
15         $S' \leftarrow$ back-propagate$(S', \eta, \partial\mathcal{L}/\partial S')$;
16 return $\Delta F_1 = F_1(D_{\text{Test}}, S') - F_1(D_{\text{Test}}, S)$;
```

Augmentation operator. The operator aug needs to be carefully designed to inject the explanations as side information for training. We achieve this goal by drawing the connection between explanations and data augmentation. Indeed, one can easily transform different formats of explanations into additional training examples for EM. We provide some examples next and they cover the various EM explainers that we will evaluate:

- For counterfactual explanations, a straightforward transformation is to add the counterfactual examples generated by E into the training set (with their labels flipped).
- For token-level explanations (e.g., the default format of LIME and SHAP) where explanations $E(B)$ assign a weight to each token, we consider a transformation that *removes* the top-p% weighted tokens from each entity entry. We expect that the removal will cause a matched pair to become non-match or vice versa.
- For attribute-level explanations such as [5] where $E(B)$ assigns weights to attributes of entity entries, we consider two transformations: attribute removal and attribute copy. Attribute removal removes the attribute of the highest weight from both the left and right entity entries. The copy operator takes the top-weighted attribute and copies the value of the attribute from the left entry to the right entry or vice versa. We expect these two operators can turn a non-match pair into a match by increasing the similarity of the entries.

We summarize the used operators in Table 1. For the token-level and attribute-level transformations, we use the teacher model T to re-label each transformed instance to ensure the correctness of its label. When multiple transformations are available, we prioritize operators that lead to counterfactual examples (i.e., with flipped labels as T predicts) and uniformly sample from all possible options.

Table 1: Transformations for each explanation type.

Exp. type	Operator(s)
Counterfactual	Add counter examples to training batch
Token-level	Remove the top-[10%, 30%, 50%, 70%] weighted tokens
Attribute-level	Remove the top weighted attr from left/right entries
	Copy the top weighted attribute from left $\longrightarrow$ right (or $\longleftarrow$)

Finally, we can return the performance difference $\Delta F_1 = F_1(D_{\text{Test}}, S') - F_1(D_{\text{Test}}, S)$ of the two rounds in F1 scores as the performance measure of the explainer E.

4.2 Experiment Setup

Next, we introduce detailed settings of the experiment including datasets, baselines, and hyper-parameters.

Datasets. We consider 5 EM datasets from the DeepMatcher repository[1]. We list the details of the datasets in Table 2. Each dataset provides ~8k to ~22k labeled pairs (Train+Valid) for training the target models. For training the student models for explanation evaluation, we split the original test sets (Test) into the training and test sets of the student models, according to the 1:1 ratio (S-Train and S-Test). Note that we do not use the rest of datasets in the DeepMatcher repository because they do not have large enough test sets to create sufficient training instances for the student models.

[1]https://github.com/anhaidgroup/deepmatcher/Datasets.md

Table 2: Statistics of EM datasets.

Datasets	Train+Valid	Test	S-Train	S-Test	%pos
Abt-Buy (AB)	7,659	1,916	958	958	10.74%
Amazon-Google (AG)	9,167	2,293	1,146	1,147	10.18%
DBLP-ACM (DA)	9,890	2,473	1,236	1,237	17.96%
DBLP-Scholar (DS)	22,965	5,742	2,871	2,871	18.63%
Walmart-Amazon (WA)	8,193	2,049	1,024	1,025	9.39%

Models to be explained. In the experiments of this paper, we choose Ditto [19] as the target EM model to be explained. Ditto is based on pre-trained Transformer models and is by far the state-of-the-art learning-based solution on the 5 EM datasets above. Transformer-based models are also known to be complex which imposes new challenges to explanation methods both in scalability and effectiveness. Specifically, we chose the default RoBERTa variant of Ditto without the data augmentation or knowledge injection optimization. There is no doubt that our work can also be applied to explain other deep learning based EM models.

We trained each model for 15 epochs with a learning rate of 3×10^{-5} with batch size 64 and chose the checkpoint of the highest F1 score on the validation set. Table 3 shows the F1 scores of the target models on the test sets. These numbers match with the original performance numbers reported in [19].

Table 3: Target models' (Ditto-RoBERTa) performance.

Dataset	AB	AG	DA	DS	WA
F1	88.89	71.40	98.66	95.46	86.56

Explainers. We report the results of 4 explanation methods applied to the target EM models:

- **LIME** [30]: LIME is a widely used method for generating black-box explanations in the format of feature importance by learning local surrogate models near the target instance. Since the original LIME does not support matching tasks like EM, we followed [5] to obtain both token weights and aggregate them into attribute weights by averaging. We output explanations in the format of top-p% tokens and top attributes.
- **SHAP** [21]: SHAP follows a similar paradigm as LIME with the additional properties that the feature weights represent "contributions" to the predictions in a Game-Theoretic principle. We follow the same pattern to extend SHAP to EM. Both LIME and SHAP perform the transformations in Table 1 to augment the student training set (S-Train) during the evaluation process.
- **CF-baseline**: We also consider baseline counterfactual explanation methods based on exhaustive search. This baseline enumerates (1) all 3-grams in the input sequence and (2) all entity attributes to see if removing the 3-gram or copying the attribute from left to right can result in a counterfactual example. It randomly selects a counterfactual example if multiple ones exist. We denote this baseline by Remove+Copy. Another baseline that only does the 3-gram removal is denoted by Remove.
- **Minun**: For our own method, we implement two variants: Minun (Greedy) is the method using a greedy algorithm to traverse the search space introduced in Section 3.2; while Minun (Binary) employs the optimization of binary search in Section 3.3.

For our own methods, we set $K = 10$ empirically. Besides, to make carnality of D_{Train} equal to that of the original training set, for each pair we only select one explanation from the K results generated by the algorithms to obtain the training instance. Specifically, we select the one with the smallest number of edit operations. If there is a tie, we will choose the one with the highest confidence score of prediction.

Since it is non-trivial to extend the counterfactual explanation approaches proposed in [12, 32] to the EM setting, we excluded them from the comparison.

Student models. We design the student model also as pre-trained Transformer models in our experiments. We choose the DistilBERT variant of Ditto. For each round of the simulation (Algorithm 3), we set the learning rate to 3×10^{-5}, batch size to 64, and the number of epochs to 40. Since there is no validation set, we select the checkpoint with the highest training F1 score and report the F1 on the S-Test set. We then follow the simulation process to report the relative improvement (ΔF_1) as the performance metrics for each evaluated explanation method. We use a less competitive EM model to make the relative improvement given by each explanation method more observable. In practice, we believe that the student model should be chosen to match the cognitive power of the target user population.

We implemented Minun and the evaluation framework using PyTorch and the Transformers library. We ran all experiments in a server machine with a configuration similar to a p4d.24xlarge AWS EC2 machine with 8 A100 GPUs.

5 RESULTS

5.1 Performance Gain of the Student Models

We follow the evaluation process and settings in Section 4 to obtain the results shown in Table 4.

We have the following observations: Firstly, the two methods LIME and SHAP adapted from explainable AI do not perform well in explaining the EM models. The average performance gains on F_1 score brought by their explanations are only 0.91 and 0.32, respectively. The reason could be that since they are not initially designed for entity matching, it is difficult for their token-level explanations to capture the pairwise matching signals within attributes, which are key to EM explanations. As a result, they provide less useful information for the student model. Indeed, as we inspect the examples generated by LIME or SHAP, only 14.1% or 13% of them are counterfactual respectively.

On the other hand, both methods of Minun achieve promising results in improving the F_1 score, which are 6.65 and 8.9 on average, respectively. The higher performance gain mainly lies on that our methods define the explanation as a transformation from one entity to another, which naturally include the pairwise matching information. Besides, Minun can provide signals from both token and attribute levels. Note that both methods fail to improve on the DA dataset. This might be caused by a characteristics of this dataset: the input entries typically follow specific formatting patterns depending on the data source (DBLP or ACM). The edit operations by Minun can violate such patterns by inserting new tokens which potentially introduce noise to the training set of the student model, while the remove and copy operators by LIME or SHAP will not cause such violations.

Finally, Minun (Binary) outperforms Minun (Greedy) in most settings. The reason might be that since Minun (Binary) traverses more parts of the search space and Minun (Greedy) tends to terminate after finding enough results locally, the explanations by Minun (Binary) are usually more diverse. Consequently, it could provide more useful information to the student model. We expect a diversified version of Minun (Greedy) to match the performance of Minun (Binary).

We also conduct experiments to compare against the two baseline counterfactual explanation approaches (see Section 2.3 and Section 4.2). Recall that Remove generates explanations by randomly removing a text span with length 3; Remove+Copy also considers copying tokens in the target attribute from one entity entry to another so as to turn the unmatched pairs into match ones. The results are shown in Table 5. We can see that the performance gains by the greedy and binary search based approaches are both significantly higher than the two baseline counterfactual approaches. This result illustrates that the generalized, finer-grained search space indeed introduces more insightful explanations for the student model by providing a richer set of augmentation operators. Defining the right search space is essential for explainable EM techniques to generate high-quality explanations.

5.2 Efficiency

Next, we evaluate the efficiency of the proposed methods. To evaluate the efficiency of each method, we look at two metrics: the average execution time per instance and the average number of calls of the teacher model per instance. An efficient explainer should be able to find the explanation in a shorter time and fewer inferences over the teacher model. The results of the average execution time are shown in Figure 4. We can see that the greedy algorithm is not so efficient due to the high computation cost in the linear scan of the candidates in the search space. Meanwhile, LIME and SHAP avoid scanning the search space since they find candidates using random sampling. With the help of binary search techniques, the efficiency of Minun is substantially improved and even outperforms LIME and SHAP. The results of the average number of calls are shown in Figure 5. We can see that LIME and SHAP require >3x more inferences over the teacher model. This means that the efficiency advantage of Minun will only be larger in a no-GPU computing environment where the inference cost will dominate. The saving of LIME and SHAP's execution time compared with Minun mainly lies in that they do not need to construct and traverse the whole search space as in Minun.

5.3 Case Study

Next, we demonstrate the effectiveness of Minun using an example from the Abt-Buy dataset. Figure 6 shows the explanations generated by LIME, SHAP, and Minun. The example consists of two product records of Canon ink cartridges. Although the two records look similar, they have different product IDs (cl52 vs. cli-8pc) and the right entry is specifically *cyan ink* cartridges. The Ditto-RoBERTa model correctly predicts this pair as an "unmatch" pair.

Table 4: Performance gains (ΔF_1 Score) of the student models (DistilBERT-Ditto) on 5 EM benchmark datasets.

Dataset	LIME			SHAP			Minun (Greedy)			Minun (Binary)		
	w/o exp	w/ exp	Δ score	w/o exp	w/ exp	Δ score	w/o exp	w/ exp	Δ score	w/o exp	w/ exp	Δ score
AG	71.16	68.15	-3.04	67.82	71.79	+3.97	63.25	69.81	+6.56	63.24	70.81	+7.57
DA	97.77	98.18	+0.41	96.25	97.75	+1.50	98.18	97.00	-1.18	98.18	97.33	-0.85
DS	93.80	94.07	+0.27	93.15	93.86	+0.71	94.22	95.86	+1.64	94.21	96.62	+2.41
WA	60.98	61.90	+0.93	58.16	54.65	-3.51	60.11	78.97	+18.86	60.11	84.11	+24.00
AB	62.86	69.77	+6.91	62.86	63.75	+0.89	72.25	79.64	+7.39	72.25	83.64	+11.39

Table 5: Comparison of ΔF_1 score with baseline counterfactual explanation methods (Remove and Remove+Copy).

Dataset	Remove	Remove+Copy	Minun (Greedy)	Minun (Binary)
AB	+2.74	-2.95	+7.3	+11.3
AG	-1.11	+0.24	+6.6	+7.6
DA	+0.31	+0.94	-1.2	-0.9
DS	+3.13	+15.22	+1.7	+24.0
WA	+3.84	+11.41	+18.8	+11.3
Average	+1.78	+4.97	+6.2	+8.9

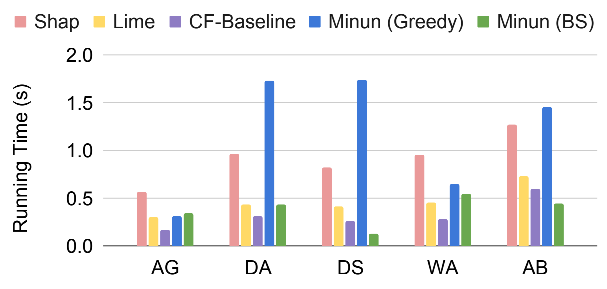


Figure 4: Results of Efficiency: Execution Time. The binary search optimization of Minun significantly improves its running time efficiency, outperforming both SHAP and LIME.

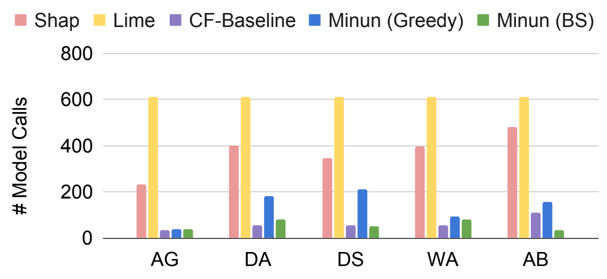


Figure 5: Results of Efficiency: Number of Calls. Minun requires significantly fewer number inference calls to the target models.

The baseline LIME and SHAP fail to generate insightful explanations for this example. With the top-30% highest weighted tokens, LIME can identify the ID span "cl52" as part of the explanation, but does not report "photo cyan" or "cyan ink" in the name/description. SHAP is able to identify useful words like "cl52", "cli-", and "cyan" in its explanations. However, tokens "cli-" and "cyan" were not shown with a higher weight threshold (i.e., at top-10%). As a result, the explanation will include tokens such as "canon", "and", and "cartridge" which are less useful. Neither explanations given by LIME or SHAP are counterfactual, that is, the target model still predicts the input pair as "unmatch" after removing those highlighted tokens.

In contrast, Minun generates a concise and accurate explanation. The counterfactual example consists of inserting the token "cli-8pc" and "cyan" from the right entry to the name attribute of the left entry. These two changes alone are sufficient to flip the target model's prediction from "not match" to "match". As such, the explanation provides enough insight that the product ID and the ink color cause the model to make a negative prediction.

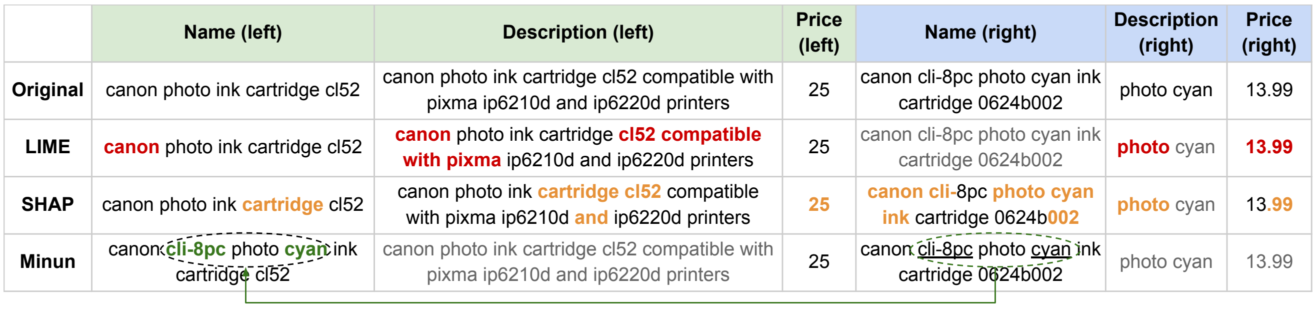
Note that both LIME and SHAP identify the price attribute as the explanation (25 vs. 13.99). Although this is a valid explanation, we found in this dataset that the price difference feature is not significant for this EM application. Only 16.7% of pairs have both non-NULL price attributes. The average price difference for the positive class is 19.4% while it is only 47.4% for the negative class. As a result, it is correct for Minun to exclude price in its explanation.

6 RELATED WORK

6.1 Entity Matching

Entity Matching (EM) is an important data integration task that has been extensively studied over the past decades [18]. There are two main steps in an EM pipeline: blocking and matching. The blocking step aims at reducing the number of potential comparisons. This is realized by generating a small candidate set while retaining as many real matches as possible. The matching step performs pairwise comparisons within each block to identify matched entities.

Many previous studies aimed at developing effective matching strategies, including rule-based and machine learning-based approaches [26]. Recently, deep learning methods have been widely adopted in EM and achieved very promising results. DeepER [9] and Deep Matcher [24] employed the Recurrent Neural Network (RNN) models to perform entity matching. Seq2SeqMatcher [25], MPM [11], and HierMatcher [10] improved the performance of matching between heterogeneous data sources by applying additional alignment layers. Some recent studies [4, 19, 28, 38] further adopted the pre-trained language models such as BERT for entity matching. Among them, Ditto [19] integrated the pre-trained language models with data augmentation techniques and achieved the state-of-the-art performance. The EM problem has also been studied by introducing different kinds of machine learning paradigms, such as probabilistic models [40], active learning [17], and meta-learning [22].



	Name (left)	Description (left)	Price (left)	Name (right)	Description (right)	Price (right)
Original	canon photo ink cartridge cl52	canon photo ink cartridge cl52 compatible with pixma ip6210d and ip6220d printers	25	canon cli-8pc photo cyan ink cartridge 0624b002	photo cyan	13.99
LIME	**canon** photo ink cartridge cl52	**canon** photo ink cartridge **cl52 compatible with pixma** ip6210d and ip6220d printers	25	canon cli-8pc photo cyan ink cartridge 0624b002	**photo** cyan	**13.99**
SHAP	canon photo ink **cartridge** cl52	canon photo ink **cartridge cl52** compatible with pixma ip6210d **and** ip6220d printers	**25**	**canon cli**-8pc **photo cyan ink** cartridge 0624b**002**	**photo** cyan	13.**99**
Minun	canon **cli-8pc** photo **cyan** ink cartridge cl52	canon photo ink cartridge cl52 compatible with pixma ip6210d and ip6220d printers	25	canon cli-8pc photo cyan ink cartridge 0624b002	photo cyan	13.99

Figure 6: An example with explanations from the Abt-Buy dataset. For LIME and SHAP, we highlight the top-30% highest weighted tokens in their explanations (in red and orange respectively). Minun generates counterfactual explanations that inserted "cli-8pc" and "cyan" to the "Name" attribute (right → left), highlighted in green. Minun discovers that inserting these two tokens causes the EM model to predict "Match" instead of "Unmatch", resulting in a concise and accurate explanation.

6.2 Explainable Entity Matching

Since deep learning has been widely adopted in EM applications, many efforts have been paid to explain the results of such DL-based approaches for EM. A high-level discussion about the design guideline is made in [35]. ExplainER [8] provided a tool to visualize the entity matching results. Mojito [5] extended the LIME method to explain the output of matching models. Landmark [2] improves Mojito by adopting a more sophisticated copy mechanism to provide more accurate and interesting explanations for entity pairs. LEMON [3] further came up with a new unit called attribution to find explanations with proper granularity. All these approaches are simple extensions of the LIME approach [30], which is designed for explaining classifiers over a single instance. For matching tasks like EM, they cannot fully leverage the rich similarity and dissimilarity signals from the textual attributes. Besides, they failed to provide an objective quantitative evaluation method and thus it is difficult to judge the quality of explanations generated by them.

6.3 Responsible Data Management

Along with the popularity of explainable AI [1], there is an increasing requirement for responsible data management [34] in many real applications. Some studies in the NLP field tried to explain the results of attention modules [33, 39] and Transformer-based language models [6, 37] with the distribution of attention weights. However, since the attention modules are built on top of several non-linear transformations, it is rather difficult to make a reasonable association between the model outcome and attention weights [16].

There has been a rich line of works that aim at providing interpretable and explainable results for black-box ML models [14]. LIME [30] proposed to use a simpler linear model to provide local explanations for black-box models. Anchors [31] generated the explanations in the format of if-else rules. SHAP [21] provided a unified Game-Theoretic framework to explain the results of multiple kinds of models and applications. The above methods focus on explaining the classifiers over a single instance and cannot be directly applied to deal with EM applications that require pairwise classification over records with multiple attributes.

Some recent studies aimed at finding counterfactual examples to explain learning-based data management applications. Geco [32] identified the nearest neighbor of counterfactual examples using a generic algorithm. Lewis [12] reached this goal by modeling the problem as causal inference. Nevertheless, these approaches (i) only support records with numerical and categorical attributes; (ii) cannot model the problem of pair classification. Thus, it is rather challenging to apply them in explaining black-box EM models.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose Minun, a model agnostic framework to provide local explanations to black-box entity matching models. We propose a novel definition of explanations based on counterfactual examples. Each explanation is formed by a sequence of modifications to the input entries such that they can flip the target EM model's original prediction. We utilize token-level edit distance between a pair of entity entries to describe the search space for counterfactual explanations and then develop two efficient algorithms to find the results. We further propose a general evaluation framework that enables quantitative evaluation of the quality of explanations by Minun or other existing explanation methods. The results of our empirical evaluation show that Minun generates high-quality explanations while being computationally more efficient.

For future work, we plan to extend Minun to a broad range of data integration (DI) tasks beyond EM. We will explore the connections between data augmentation and explanations. More specifically, we would like to generalize the search space definition of Minun by a declarative framework for specifying data transformations that can be interpreted as explanations, e.g., deleting certain informative tokens, replacement of an ID span, applying a data dependency, etc. Next, since we can now evaluate the quality of explanations by augmenting the student model's training data, we can formulate the task of finding explanations as a task of finding *effective data augmentation policies*. It is possible that we can use the signal from the student model to guide the search of the most effective transformations, using a meta-learning framework similar to Rotom [22]. We expect this new framework can support explanations of a variety of data integration and preparation tasks including entity matching, information extraction, and data cleaning.

REFERENCES

[1] A. B. Arrieta, N. D. Rodríguez, J. D. Ser, and et al. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020.

[2] A. Baraldi, F. D. Buono, M. Paganelli, and F. Guerra. Using landmarks for explaining entity matching models. In *EDBT*, pages 451–456, 2021.

[3] N. Barlaug. LEMON: explainable entity matching. *CoRR*, abs/2110.00516, 2021.

[4] U. Brunner and K. Stockinger. Entity matching with transformer architectures - A step forward in data integration. In A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang, editors, *EDBT*, pages 463–473, 2020.

[5] V. D. Cicco, D. Firmani, N. Koudas, P. Merialdo, and D. Srivastava. Interpreting deep learning models for entity resolution: an experience report using LIME. In *aiDM@SIGMOD*, pages 8:1–8:4, 2019.

[6] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What does BERT look at? an analysis of bert's attention. In *BlackboxNLP@ACL 2019*, pages 276–286, 2019.

[7] F. Doshi-Velez and B. Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017.

[8] A. Ebaid, S. Thirumuruganathan, W. G. Aref, A. K. Elmagarmid, and M. Ouzzani. EXPLAINER: entity resolution explanations. In *ICDE*, pages 2000–2003, 2019.

[9] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.

[10] C. Fu, X. Han, J. He, and L. Sun. Hierarchical matching network for heterogeneous entity resolution. In C. Bessiere, editor, *IJCAI*, pages 3665–3671, 2020.

[11] C. Fu, X. Han, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong. End-to-end multi-perspective matching for entity resolution. In S. Kraus, editor, *IJCAI*, pages 4961–4967, 2019.

[12] S. Galhotra, R. Pradhan, and B. Salimi. Explaining black-box algorithms using probabilistic contrastive counterfactuals. In *SIGMOD*, pages 577–590, 2021.

[13] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[14] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019.

[15] P. Hase and M. Bansal. Evaluating explainable AI: which algorithmic explanations help users predict model behavior? In *ACL*, pages 5540–5552, 2020.

[16] S. Jain and B. C. Wallace. Attention is not explanation. In *NAACL-HLT*, pages 3543–3556, 2019.

[17] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa. Low-resource deep entity resolution with transfer and active learning. In *ACL*, pages 5851–5861, 2019.

[18] P. Konda, S. Das, P. S. G. C., A. Doan, and et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[19] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.*, 14(1):50–60, 2020.

[20] Y. Li, J. Li, Y. Suhara, J. Wang, W. Hirota, and W. Tan. Deep entity matching: Challenges and opportunities. *ACM J. Data Inf. Qual.*, 13(1):1:1–1:17, 2021.

[21] S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017.

[22] Z. Miao, Y. Li, and X. Wang. Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond. In *SIGMOD*, pages 1303–1316, 2021.

[23] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[24] 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 *SIGMOD*, pages 19–34, 2018.

[25] H. Nie, X. Han, B. He, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong. Deep sequence-to-sequence entity matching for heterogeneous entity resolution. In *CIKM*, pages 629–638, 2019.

[26] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2):31:1–31:42, 2020.

[27] R. Peeters and C. Bizer. Dual-objective fine-tuning of BERT for entity matching. *Proc. VLDB Endow.*, 14(10):1913–1921, 2021.

[28] R. Peeters, C. Bizer, and G. Glavas. Intermediate training of BERT for product matching. In F. Piai, D. Firmani, V. Crescenzi, A. D. Angelis, X. L. Dong, M. Mazzei, P. Merialdo, and D. Srivastava, editors, *DI2KG@VLDB*, 2020.

[29] D. Pruthi, B. Dhingra, L. B. Soares, M. Collins, Z. C. Lipton, G. Neubig, and W. W. Cohen. Evaluating explanations: How much do explanations from the teacher aid students? *CoRR*, abs/2012.00893, 2020.

[30] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *ACM SIGKDD*, pages 1135–1144, 2016.

[31] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, pages 1527–1535, 2018.

[32] M. Schleich, Z. Geng, Y. Zhang, and D. Suciu. Geco: Quality counterfactual explanations in real time. *Proc. VLDB Endow.*, 14(9):1681–1693, 2021.

[33] S. Serrano and N. A. Smith. Is attention interpretable? In *ACL*, pages 2931–2951, 2019.

[34] J. Stoyanovich, B. Howe, and H. V. Jagadish. Responsible data management. *PVLDB*, 13(12):3474–3488, 2020.

[35] S. Thirumuruganathan, M. Ouzzani, and N. Tang. Explaining entity resolution predictions: Where are we and what needs to be done? In *HILDA@SIGMOD*, pages 10:1–10:6, 2019.

[36] S. Thirumuruganathan, N. Tang, M. Ouzzani, and A. Doan. Data curation with deep learning. In *EDBT*, pages 277–286, 2020.

[37] B. van Aken, B. Winter, A. Löser, and F. A. Gers. How does BERT answer questions?: A layer-wise analysis of transformer representations. In *CIKM*, pages 1823–1832, 2019.

[38] J. Wang, Y. Li, and W. Hirota. Machamp: A generalized entity matching benchmark. In *CIKM*, pages 4633–4642. ACM, 2021.

[39] S. Wiegreffe and Y. Pinter. Attention is not not explanation. In *EMNLP-IJCNLP*, pages 11–20, 2019.

[40] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In *SIGMOD*, pages 1149–1164, 2020.