A Critical Re-evaluation of Record Linkage Benchmarks for Learning-Based Matching Algorithms

George Papadakis
National & Kapodistrian University of Athens, Greece
gpapadis@di.uoa.gr

Nishadi Kirielle
The Australian National University, Australia
nishadi.kirielle@anu.edu.au

Peter Christen
The Australian National University, Australia
peter.christen@anu.edu.au

Themis Palpanas
Université Paris Cité & French University Institute, France
themis@mi.parisdescartes.fr

***Abstract*—Entity resolution (ER) is the process of identifying records that refer to the same entities within one or across multiple databases. Numerous techniques have been developed to tackle ER challenges over the years, with recent emphasis placed on machine and deep learning methods for the matching phase. However, the quality of the benchmark datasets typically used in the experimental evaluations of learning-based matching algorithms has not been examined in the literature. To cover this gap, we propose four complementary approaches to assessing the difficulty and appropriateness of 13 commonly used datasets: two theoretical ones, which involve new measures of linearity and existing measures of complexity, and two practical ones – the difference between the best non-linear and linear matchers, as well as the difference between the best learning-based matcher and the perfect oracle. Our analysis demonstrates that most existing benchmark datasets pose rather easy classification tasks. As a result, they are not suitable for properly evaluating learning-based matching algorithms. To address this issue, we propose a new methodology for yielding benchmark datasets. We put it into practice by creating four new matching tasks, and we verify that these new benchmarks are more challenging and therefore more suitable for further advancements in the field.**

***Index Terms*—Record Linkage, Supervised Matching, Deep Learning, Benchmark Analysis**

I. Introduction

Entity Resolution (ER) aims to identify and link records that refer to the same entity across databases, called *duplicates* [1]. ER has been an active topic of research since the 1950s [2], while various learning-based ER techniques, both supervised and unsupervised, have been developed in the past two decades. For overviews of ER see [3], [4], [5], [6].

ER faces several major challenges. First, databases typically contain no unique global entity identifiers that would allow an exact join to identify those records that refer to the same entities. As a result, *matching* methods compare quasi-identifiers (QIDs) [4], such as names and addresses of people, or titles and authors of publications. The assumption here is that the more similar their QIDs are, the more likely the corresponding records are to be matching. Second, as databases are getting larger, comparing all possible pairs of records is infeasible, due to the quadratic cost. Instead, *blocking*, *indexing*, or *filtering* techniques [7] typically identify the candidate pairs or groups of records that are forwarded to matching.

In recent years, a diverse range of methods based on machine learning (ML) [8] and especially deep learning (DL) has been developed to address the first challenge, namely matching [9], [10]. Due to the similarity of ER to natural language processing tasks, such as machine translation or entity extraction and recognition, many DL-based matching techniques leverage relevant technologies like pre-trained language models. The experimental results reported have been outstanding, as these methods maximize matching effectiveness in many benchmark datasets [10], [11], [12].

However, the quality of these benchmark datasets has been overlooked in the literature – the sole exception is the analysis of the large portion of entities shared by training and testing sets, which results in low performance in the case of unseen test entities [13]. Existing ER benchmark datasets typically treat matching as a binary classification task that applies to a set of candidate pairs generated after blocking, which typically has a significant impact on the resulting performance [5], [14]. In general, a loose blocking approach achieves high recall, ensuring that all positive instances (i.e., matching pairs) are included, at the cost of many negative ones (i.e., non-matching pairs) with low similarity, which can thus be easily discarded by a learning-based matching algorithm, even a linear one. In contrast, a strict blocking approach might sacrifice a small part of the positive instances, but mostly includes highly similar negative ones, which involve nearest neighbors and are harder to classify, thus requiring more effective, complex and non-linear learning-based matching algorithms. Nevertheless, most existing datasets lack any documentation about the blocking process that generated their candidate pairs, i.e., no information is provided about which blocking method was used, how it was configured and which attributes provided the textual evidence for creating blocks. As a result, there is a large deviation in core characteristics like the imbalance ratio between the existing benchmarks and those created through a principled approach that employs a fine-tuned state-of-the-art blocking method, as we describe in Section VI.

In this paper, we aim to cover the above gap by proposing a principled framework for assessing the quality of benchmark datasets for learning-based matching algorithms. It consists of two *complementary* types of measures. First, *a-priori* measures theoretically estimate the appropriateness of a benchmark dataset, based on the characteristics of its classes. We propose novel measures that estimate the degree of linearity in a benchmark dataset as well as existing complexity measures that are applied to ER benchmarks for the first time. Second, *a-posteriori* measures rely on the performance of matching algorithms on the data at hand.

To put the latter measures into practice, we consider 7 open-source, non-linear ML- and DL-based matching algorithms, which include the state-of-the-art techniques in the field. We complement them with 6 novel matching algorithms, which perform linear classification, thus estimating the baseline performance of learning-based methods. These two types of algorithms allow for estimating the real advantage of non-linear learning-based matching algorithms over simple linear ones, and their distance from the ideal matcher (the perfect oracle).

When applying our a-priori and a-posteriori measures on widely used benchmark ER datasets, our experimental evaluation shows that most of these datasets are inappropriate for evaluating the full potential of complex matchers, such as DL-based ones. To address this issue, we propose a novel way of constructing benchmarks from the same original datasets based on blocking and the knowledge of the complete set of real true duplicates. We apply all our measures to these new benchmarks, demonstrating that they form harder classification tasks that highlight the advantages of complex matchers.

To the best of our knowledge, these topics have not been examined in the literature before. We make the following contributions: (1) In Section III, we coin novel theoretical measures for *a-priori* assessing the difficulty of ER benchmark datasets. We also introduce two novel aggregate measures that leverage a series of matching algorithms to *a-posteriori* assess the difficulty of ER benchmarks. (2) In Section IV, we introduce a taxonomy of DL-based matching methods that facilitates the understanding of their functionality, showing that we consider a representative sample of the recent developments in the field. We also define a new family of linear learning-based matching algorithms, whose performance depends heavily on the difficulty of ER benchmarks. These algorithms lay the ground for estimating the two a-posteriori measures. (3) In Section V, we perform the first systematic evaluation of 13 popular ER benchmarks, demonstrating experimentally that most of them are too easy to classify to properly assess the performance of matching algorithms in real ER scenarios. (4) In Section VI, we propose a novel methodology for creating new ER benchmarks and experimentally demonstrate that they are more suitable for assessing the benefits of matchers. All our code and data are publicly available.[1]

II. Problem Definition

The goal of ER is to identify duplicates, i.e., different records that describe the same real-world entities. To this end, an ER matching algorithm receives as input a set of candidate record pairs C. These are likely matches that are produced by a blocking or filtering technique [7], which is used to reduce the inherently quadratic computational cost (instead of considering all possible pairs, it restricts the search space to highly similar pairs only). For each record pair $(r_i, r_j) \in C$, a matching algorithm decides whether $r_i \equiv r_j$ or not, where $\equiv$ indicates that they are duplicates (refer to the same entity). The resulting set of matching pairs is denoted by M, and the non-matching pairs by N (where $C = M \cup N$ and $M \cap N = \emptyset$).

This task naturally lends itself to a binary classification setting. In this case, C constitutes the testing set, which is accompanied by a training and a validation set, T and V, respectively, with record pairs having known labels, such that C, T and V are mutually exclusive. As a result, the performance of matching is typically assessed through the *F-Measure* ($F1$) [15], which is the harmonic mean of recall (Re) and precision (Pr), i.e., $F1 = 2 \cdot Re \cdot Pr/(Re + Pr)$, where Re expresses the portion of existing duplicates that are classified as such, i.e., $Re = |G \cap M|/|G|$ with G denoting the ground truth (the set of true duplicates), and Pr determines the portion of detected matches that correspond to duplicates, i.e., $Pr = |G \cap M|/|M|$ [16], [17]. All these measures are defined in $[0, 1]$, with higher values indicating higher effectiveness.

In this given context, we formally define matching as:

Problem 1 (Matching): Given a testing set of candidate pairs C along with a training and a validation set, T and V, respectively, such that $C \cap T = \emptyset$, $C \cap V = \emptyset$, and $T \cap V = \emptyset$, train a binary classification model that splits the elements of C into the set of matching and non-matching pairs, M and N respectively, such that $F1$ is maximized.

By considering the candidate pairs from blocking, this definition is generic enough to cover any type of ER. Following the literature on learning-based matching algorithms, we exclusively consider record linkage (RL) [4], also known as *Clean-Clean* ER [18], where the input involves two individually duplicate-free, but overlapping databases [12], [19], [20].

III. Measures of Difficulty

We now describe two types of theoretical measures and two practical ones for a-priori and a-posteriori assessing the difficulty of ER benchmark datasets. All operate in a *schema-agnostic* manner that considers all attribute values in every record, disregarding the attribute structures in the given data sources. In preliminary experiments, we also explored schema-aware settings, applying the same measures to specific attribute values. These settings, though, showed no significant difference in performance in comparison to the schema-agnostic settings for both types of theoretical measures. Thus, we omit them for brevity, but report them in the extended version [21].

A. Degree of Linearity

To assess the difficulty of matching benchmarks, we introduce two new measures for estimating the success of a *linear classifier*. The higher their scores are, the easier it is for *any* supervised matching algorithm to achieve high effectiveness on the corresponding dataset. This means that

[1]See https://github.com/gpapadis/DLMatchers for more details.

Algorithm 1: Estimating the degree of linearity

```
Input: Training, validation, and testing sets $T, V, C$,
       respectively, and the similarity measure $sim$
Output: Linearity degree $F1^{max}_{sim}$ and best threshold $t_{best}$
1  $S \leftarrow \{\}$, $D \leftarrow T \cup V \cup C$
2  foreach $(r_i, r_j) \in D$ do
3      $T_i \leftarrow tokens(r_i)$, $T_j \leftarrow tokens(r_j)$
4      $S \leftarrow S \cup (sim(T_i, T_j), (r_i, r_j))$
5  $F1^{max}_{sim} \leftarrow 0$, $t_{best} \leftarrow 0$
6  for $t \leftarrow 0.01$ to $0.99$ by $0.01$ do
7      $M \leftarrow \{\}$, $N \leftarrow \{\}$
8      foreach $(s_{i,j}, (r_i, r_j)) \in S$ do
9          if $t \leq s_{i,j}$ then $M \leftarrow M \cup \{(r_i, r_j)\}$ // Match
10         else $N \leftarrow N \cup \{(r_i, r_j)\}$ // Non-match
11     if $F1^{max}_{sim} < F1(M, N, D)$ then
12         $F1^{max}_{sim} \leftarrow F1(M, N, D)$, $t_{best} \leftarrow t$
13 return $F1^{max}_{sim}$, $t_{best}$
```

benchmarks with a high degree of linearity are not suitable for highlighting the differences between complex, non-linear classifiers like those leveraging deep learning in Section IV-A. Instead, datasets with a low degree of linearity are more likely to stress the pros and cons of different classifiers.

In this context, we propose Algorithm 1, which relies on all labels in a benchmark dataset. First, it merges the training with the validation and the testing sets into a single dataset D in line 1. Then, for every candidate pair $(r_i, r_j) \in D$, it creates two token sequences, T_i and T_j, where T_x comprises the set of tokens in all attribute values in record r_x, after converting all tokens to lower-case (lines 2 and 3). A similarity score per pair, $s_{i,j} = sim(T_i, T_j) \in [0, 1]$, is then calculated based on T_i and T_j and added to the set of similarities S along with the candidate pair in line 4. Finally, the algorithm classifies all labelled pairs using a threshold t with the following rule: if $t \leq s_{i,j}$, we have a matching pair (line 9) otherwise a non-matching pair (line 10). In line 6, the algorithm loops over all thresholds in $[0.01, 0.99]$ with an increment of 0.01, and identifies the threshold that results in the highest F-measure value (lines 11 and 12). We denote this maximum F1 as the degree of linearity, $F1^{max}_{sim}$, which is returned as output along with the corresponding threshold, t_{best} in line 13.

We consider two similarity measures between the token sequences T_i and T_j of the candidate pair $c_{i,j} = (r_i, r_j)$:

1) The Cosine similarity: $CS(c_{i,j}) = |T_i \cap T_j| / \sqrt{|T_i| \times |T_j|}$.
2) The Jaccard similarity: $JS(c_{i,j}) = |T_i \cap T_j| / |T_i \cup T_j|$.

They yield two degrees of linearity: $\mathbf{F1^{max}_{CS}}$ and $\mathbf{F1^{max}_{JS}}$, respectively. By considering their maximum possible value, these measures indicate the optimal performance of a linear matching algorithm, with F1=1.0 indicating perfect separation (no false matches and no false non-matches). Other measures like the Dice or Overlap similarities [16] also seem applicable, but are linearly dependent on the Cosine and Jaccard similarities, thus providing no additional useful information.

TABLE I
DEFINITION OF COMPLEXITY MEASURES

f_1	Maximum Fisher's discriminant ratio
f_{1v}	Directional-vector maximum Fisher's discriminant ratio
f_2	Volume of the overlapping region
f_3	Max individual feature efficiency in separating the classes

(a) Feature-based measures

l_1	Sum of the error distance by linear programming
l_2	Error rate of a linear SVM classifier

(b) Linearity measures

n_1	Fraction of borderline points
n_2	Ratio of intra/extra class nearest neighbor distance
n_3	Error rate of the nearest neighbor classifier
n_4	Non-linearity of the nearest neighbor classifier
t_1	Fraction of hyperspheres covering data
lsc	Local set average cardinality

(c) Neighborhood measures

den	Average density of the network
cls	Clustering coefficient
hub	Hub score

(d) Network measures

c_1	Entropy of class proportions
c_2	Imbalance ratio

(e) Class balance measures

B. Complexity Measures

Measures for estimating the complexity of imbalanced classification tasks are examined in [22], [23]. They serve the same purpose as the degree of linearity, determining whether a benchmark is suitable for comparing learning-based matching algorithms. In this case, the lower the average score of a dataset is, the easier is the corresponding classification task. Collectively, these measures consider versatile and comprehensive evidence that is complementary to the degree of linearity; there are datasets where the degree of linearity is low, but the average complexity score suggests otherwise, indicating that simple patterns suffice for a high effectiveness, and vice versa.

Essentially, there are five types of such measures, as shown in Table I, which summarizes them: (a) The *feature-based measures* assess how discriminative the numeric features are. (b) The *linearity measures* check how effective the hyperplane defined by a linear SVM classifier is in separating the two classes. (c) The *neighborhood measures* characterize the decision boundary between the two classes, taking into account the class overlap in local neighborhoods according to the Gower distance [24]. (d) The *network measures* model a dataset as a graph, whose nodes correspond to instances and the edges connect pairs of instances with a Gower distance [24] lower than a threshold. Edges between matching and non-matching instances are pruned after constructing the graph. (e) The *class balance measures* rely on the relative class sizes.

These 17 measures yield values in $[0, 1]$, with higher values indicating more complex classification tasks. They assume that every instance is represented as a set of features suitable for the data at hand. In our case, every instance corresponds to a candidate pair of entities, $c_{i,j} \in D$. To transform a pair into a feature vector, we use the same methodology as in Section III-A, modelling $c_{i,j}$ as $f_{i,j} = [CS(c_{i,j}), JS(c_{i,j})]$, where $CS(c_{i,j})$ and $JS(c_{i,j})$ are defined above.

In this context, the dimensionality measures defined in [22], [23] do not provide any useful information and can be excluded from our analysis: t_2, t_3 and t_4 remain constant at 0, 0 and 0.5 in all cases, because we use just two features per instance. We also exclude f_4 from the feature-based measures and l_3 from the linearity ones, because in each dataset they are almost identical with f_3 and l_2, respectively.

C. Practical Measures

The above a-priori measures provide no evidence about the actual performance of learning-based matching algorithms on a particular benchmark. To cover this aspect, we complement them with two a-posteriori measures that encapsulate the performance of the matching algorithms in Section V-B. These measures help to identify benchmarks that contain a considerable portion of non-linearly separable candidate pairs, thus yielding low scores for the a-priori measures, but are still not suitable for benchmarking matching algorithms. There are two conditions for these cases: (i) a linear matching algorithm achieves a performance comparable to the top-performing non-linear ones, and (ii) the maximum F1 score among all learning-based matching algorithms is very close to the maximum possible score of F1=1. Only datasets satisfying none of these conditions are suitable for benchmarking supervised matching algorithms, despite their low a-priori scores.

Our practical measures include two ML-based and five DL-based matching algorithms, each combined with different configurations, as we describe in more detail in Section V-B. Overall, we consider 7 state-of-the-art algorithms, which together provide a representative performance of non-linear, learning-based techniques. Especially the DL-based algorithms cover all subcategories in our taxonomy, as shown in Table II. Along with the above six linear classifiers, they yield two novel, aggregate measures for assessing the advantage of the non-linear and the potential of all learning-based matchers:

1) *Non-linear boost* (NLB) is defined as the difference between the maximum F1 of all considered ML- and DL-based matching algorithms and the maximum F1 of all linear ones. The larger its value is, the greater is the advantage of non-linear classifiers, due to the high difficulty of an ER benchmark. In contrast, values close to zero indicate trivial ER benchmarks with linearly separable classes.
2) *Learning-based margin* (LBM) is defined as the difference between 1 and the maximum F1 of all considered learning-based matching algorithms. The higher its value is for a benchmark, the more room for improvements there is. Low values, close to zero, indicate datasets where learning-based matchers already exhibit practically perfect performance.

IV. Matching Algorithms

To quantify the practical measures, we use three types of matchers. Each one is presented in a different subsection.

A. DL-based Matching Algorithms

Selection Criteria. We consider as many DL-based matching algorithms as possible to get a reliable estimation on this type of algorithms on each dataset. To this end, we consider algorithms that satisfy the following four selection criteria:

(1) *Publicly available implementation:* All DL-based algorithms involve hyperparameters that affect their performance to a large extent, but for brevity or due to limited space, their description and fine-tuning are typically omitted in the context of a scientific publication. Reproducing experiments can therefore be a challenging task that might bias the results of our experimental analysis. In fact, as our experimental results in Section V-B demonstrate, it is also challenging to reproduce the performance of publicly available matching algorithms. To avoid such issues, we exclusively consider methods with a publicly released implementation.

(2) *No auxiliary data sources:* Practically, all DL-based matching algorithms leverage deep neural networks in combination with pre-trained language models, such as fastText [25] or BERT-based models [26], [27], which transform every input record into a (dense) numerical vector. Despite the different sources of embedding vectors, this approach is common to all methods we analyze, ensuring a fair comparison. We exclude, though, any other external evidence, like a knowledge-base that could be used for transfer learning [28], [29].

(3) *Scope:* We exclusively consider RL, excluding methods for multi-source ER [30] and for entity alignment [31], [32].

(4) *Guidelines:* We exclude open-source algorithms that provide neither instructions nor examples on how to use them (despite contacting their authors).

Due to the first criterion, we exclude well-known techniques like *Seq2SeqMatcher* [33], *GraphER* [34], *CorDEL* [35], *EmbDI* [36] (despite contacting its authors) and *Leva* [37]. The second criterion excludes DL-based methods that aim to reduce the size of the training set through transfer and active learning approaches, such as *Auto-EM* [28], *DeepMatcher+* [29], *DIAL* [38] and *DADER* [39]. The third criterion leaves out methods on tasks other than matching, like *Name2Vec* [40] and *Auto-ML* [41], methods crafted for multi-source ER like *JointBERT* [30], as well as all DL-based methods targeting the entity alignment problem [31], [42] (these require non-trivial adaptations for matching). The fourth criterion excludes *MCAN* [43] and *HIF-KAT* [44], as we could not run their code. Finally, we exclude *DeepER* [45], since it is subsumed by DeepMatcher [10], as explained below.

Taxonomy. To facilitate a better understanding of the DL-based matching algorithms, we propose a new taxonomy that is formed by the following three dimensions:

1) *Language model type:* We distinguish methods as being *static* or *dynamic*. The former leverage pre-trained language models that associate every token with the same embedding vector, regardless of its context. Methods such as word2vec [46], [47], Glove [48] and fastText [25] fall into this category. The opposite is true for dynamic methods, which leverage context-aware BERT-based language models [26], [27], [49], [50]. Based on the context of every token, they support polysemy, where the same word has different meanings (e.g., 'bank' as an institution and 'bank'

TABLE II
TAXONOMY OF THE SELECTED DL-BASED ER METHODS.

DL-based algorithm	Token embedding context	Schema awareness	Entity similarity context
DeepMatcher	Static	Homogeneous	Local
EMTransformer	Dynamic	Heterogeneous	Local
GNEM	Static, Dynamic	Homogeneous	Global
DITTO	Dynamic	Heterogeneous	Local
HierMatcher	Dynamic	Heterogeneous	Local

as the edge of a river) as well as synonymy, where different words have similar meanings (e.g., 'job' and 'profession').

2) *Schema awareness:* We distinguish methods as being *homogeneous* and *heterogeneous*. The former require that both input databases have the same or at least aligned schemata, unlike methods in the latter category.
3) *Entity similarity context.* We distinguish methods as being *local* and *global*. The former receive as input the textual description of two entities and decide whether they are matching or not, based exclusively on their encoding and the ensuing similarity. Global methods leverage contextual information, which goes beyond the semantic similarity of record pairs, e.g., by leveraging knowledge from the entire input datasets (e.g., overall term salience), or from the relations between candidates.

Regarding the first dimension, we should stress that the dynamic approaches cast matching as a *sequence-pair classification problem*. All QID attribute values in a record are concatenated into a single string representation called *sequence*. Then, every candidate pair is converted into the following string representation that forms the input to the neural classifier: "$[CLS]$ `Sequence 1` $[SEP]$ `Sequence 2` $[SEP]$", where $[CLS]$ and $[SEP]$ are special tokens that designate the beginning of a new candidate pair and the end of each entity description, respectively. In practice, every input should involve up to 512 tokens, which is the maximal attention span of transformer models [12]. These methods also require fine-tuning on a task-specific training set, while their generated vectors are much larger than those of the static ones (768 versus 300 dimensions [45], [51]).

Table II shows that the considered DL-based matching algorithms cover *all* types defined by our taxonomy, providing a representative sample of the field.

Methods Overview. We describe the five DL-based methods satisfying our selection criteria in chronological order.

DeepMatcher [10] (Jun. 2018) proposes a framework for DL-based matching algorithms that generalizes *DeepER* [45]. by combining three modules: (1) The attribute embedding one converts every word of an attribute value into a static embedding vector using an existing pre-trained model, such as fastText [25]. (2) The attribute similarity vector module operates in a homogeneous way that summarizes the sequence of token embeddings in each attribute and then obtains a similarity vector between every pair of candidate records (local functionality). (3) The classification module employs a two-layer fully connected ReLU HighwayNet [52], followed by a softmax layer for classification.

EMTransformer [12] (Mar. 2020) employs dynamic token embeddings using attention-based transformer models like BERT [53]. The selected model is applied in an out-of-the-box manner. To handle noise, especially in the form of misplaced attribute values (e.g., name associated with profession), it concatenates all attribute values per entity and then vectorizes them (heterogeneous approach). Every pair of records is processed independently of all others (local operation).

GNEM [19] (Apr. 2020) is a global approach that considers the relations between all candidate pairs that result from blocking. For the semantic similarity of record pairs, it leverages static (e.g., fastText) or dynamic (e.g., BERT) language models. Its interaction module simultaneously calculates the matching likelihood between all candidate pairs. It assumes that all input records are described by the same schema, therefore involving a homogeneous operation.

DITTO [11], [20] (Sep. 2020) extends EMTransformer's straightforward application of dynamic BERT-based models in three ways: (1) It incorporates domain knowledge through a named entity recognition model that identifies entity types (such as persons or dates), as well as regular expressions for identities (like product ids). (2) Its heterogeneous functionality summarizes long attribute values that exceed the 512-tokens limit of BERT by keeping only the tokens that do not correspond to stop-words and have a high TF-IDF weight. (3) It uses data augmentation to artificially produce additional training instances. It processes every candidate pair independently of the others in a local manner.

HierMatcher [54] (Jan. 2021) constructs a hierarchical neural network with four layers: (1) The *representation layer* is a dynamic approach that leverages static embeddings. (2) The *token matching layer* performs a cross-attribute token alignment that renders HierMatcher a heterogeneous approach. (3) The *attribute matching layer* adjusts the contribution of every token in an attribute value according to its importance. (4) The *entity matching layer* builds a local comparison vector by concatenating the outcomes of the previous layer.

B. Non-neural, Non-linear ML-based Methods

Two ML-based methods are typically used in the literature, both of which satisfy the selection criteria in Section IV-A.

Magellan [8] (Sep. 2016) combines traditional ML classifiers with a set of automatically extracted features: every candidate pair is mapped to a numerical feature vector, where every dimension corresponds to the score of a particular similarity function on a specific attribute. Magellan implements several established functions like Jaro, Jaccard, Monge-Elkan etc [16]. Several established classifiers are also supported.

ZeroER [55] (Jun. 2020) is an unsupervised method that uses the same features as Magellan, but requires no training data. It extends the expectation-maximisation approach [56] through Gaussian mixture models that capture the distributions of matches and non-matches, while considering the dependencies between different features.

C. Non-neural, Linear Supervised Methods

We now present our new linear matching algorithms for efficiently assessing the difficulty of a benchmark dataset. They are inspired from the degree of linearity, but unlike Algorithm 1, they do not report a characteristic of the dataset based on the tokens in the entity profiles of all labeled instances. They use only the training and validation sets for their learning, while being more flexible, using character n-grams and language models for the similarity estimation between each pair of profiles (not just tokens). Their performance can be measured in terms of effectiveness, time and space complexity, thus being suitable for a holistic comparison with the matching algorithms proposed in the literature.

Algorithm 2 details our methods. For each pair of records in the training set T, (r_i, r_j), it extracts the features from their QID attributes, e.g., by tokenizing all the values on whitespaces (lines 2 and 3). Using the individual features, the algorithm calculates and stores the feature vector $f_{i,j}$ for each record pair (r_i, r_j) in lines 4 and 5. For example, $f_{i,j}$ could be formed by comparing the tokens of the individual features with similarity measures like Cosine, Jaccard and Dice. Note that all dimensions in $f_{i,j}$ are defined in $[0, 1]$.

Next, the algorithm identifies the threshold that achieves the highest F1 score per feature, when applied to the instances of the training set. Line 6 loops over all thresholds in $[0.01, 0.99]$ with a step increment of 0.01, and lines 7 to 11 generate a set of matching and non-matching pairs for each individual feature f. Using these sets, lines 12 to 14 estimate the maximum F1 score per feature f, $F1^T_{max}[f]$, and the respective threshold.

The resulting configurations are then applied in lines 15 to 24 to the validation set in order to identify the overall best feature and the respective threshold. For each candidate pair in V (line 16), the algorithm extracts the individual features and the corresponding feature vector (lines 17 and 18), using the same functions that were applied to the training instances in lines 3 and 4. Rather than storing the feature vectors, it directly applies the best threshold per feature to distinguish the matching from the non-matching pairs of each dimension (lines 19 to 21). This lays the ground for estimating $F1$ per feature and the maximum $F1$ over all features in lines 22 to 24.

Finally, the algorithm applies the identified best feature and threshold to the testing instances in order to estimate the overall $F1$ in lines 25 to 29. For each candidate pair in C, it estimates only the value of the feature with the highest performance over the validation set (lines 26 and 27). Using the respective threshold, it distinguishes the matching from the non-matching pairs and returns the ensuing $F1$ (lines 28-30).

We call this algorithm *Efficient Supervised Difficulty Estimation* (**ESDE**). Its space complexity is linear, storing one feature vector per training instance. Its time complexity is linear, going through the training set 100 times, $|F|$ times through the validation set (where $|F|$ denotes the dimensionality of the feature vector and is practically constant in each dataset, as explained below), and once through the testing set. It is also versatile, accommodating diverse feature vectors:

Algorithm 2: Efficient Supervised Difficulty Estimation

```
Input: Training $T$, validation $V$ and testing $C$ sets & the set of
       features $F$
Output: The F-Measure on the testing set $F1$
1  $F_T \leftarrow \{\}$, $F1^T_{max}[] \leftarrow \{\}$, $t^T_{best}[] \leftarrow \{\}$
2  foreach $(r_i, r_j) \in T$ do // Training phase
3      $F_i \leftarrow getFeatures(r_i)$, $F_j \leftarrow getFeatures(r_j)$
4      $f_{i,j} \leftarrow getFeatureVector(F_i, F_j)$
5      $F_T \leftarrow F_T \cup (f_{i,j}, (r_i, r_j))$
6  for $t \leftarrow 0.01$ to $0.99$ by $0.01$ do
7      $M_F[] \leftarrow \{\}$, $N_F[] \leftarrow \{\}$
8      foreach $(f_{i,j}, (r_i, r_j)) \in F_T$ do
9          foreach $f \in f_{i,j}$ do
10             if $t \leq f$ then $M[f] \leftarrow M[f] \cup \{(r_i, r_j)\}$
11             else $N[f] \leftarrow N[f] \cup \{(r_i, r_j)\}$
12     foreach $f \in F$ do
13         if $F1^T_{max}[f] < F1(M[f], N[f], T)$ then
14             $F1^T_{max}[f] \leftarrow F1(M[f], N[f], T)$, $t^T_{best}[f] \leftarrow t$
15 $F1^V_{max} \leftarrow 0$, $f_{best} \leftarrow null$, $t_{best} \leftarrow 0$, $M_F[] \leftarrow \{\}$, $N_F[] \leftarrow \{\}$
16 foreach $(r_i, r_j) \in V$ do // Validation phase
17     $F_i \leftarrow getFeatures(r_i)$, $F_j \leftarrow getFeatures(r_j)$
18     $f_{i,j} \leftarrow getFeatureVector(F_i, F_j)$
19     foreach $f \in f_{i,j}$ do
20         if $t^T_{best}[f] \leq f$ then $M[f] \leftarrow M[f] \cup \{(r_i, r_j)\}$
21         else $N[f] \leftarrow N[f] \cup \{(r_i, r_j)\}$
22 foreach $f \in F$ do
23     if $F1^V_{max} < F1(M[f], N[f], V)$ then
24         $F1^V_{max} \leftarrow F1(M[f], N[f], V)$, $f_{best} \leftarrow f$,
           $t_{best} \leftarrow t^T_{best}[f]$
25 $M \leftarrow \{\}$, $N \leftarrow \{\}$
26 foreach $(r_i, r_j) \in C$ do // Testing phase
27     $s_{best} \leftarrow getFeature(r_i, r_j, f_{best})$
28     if $t_{best} \leq s_{best}$ then $M \leftarrow M \cup \{(r_i, r_j)\}$
29     else $N \leftarrow N \cup \{(r_i, r_j)\}$
30 return $F1(M, N, C)$
```

1) *Schema-agnostic ESDE* (**SA-ESDE**). $getFeatures()$ represents every record r_i by its distinct tokens, F_i=T_i. Every candidate pair $c_{i,j}$ is represented by the vector $f_{i,j} = [CS_{i,j}, DS_{i,j}, JS_{i,j}]$, where $CS_{i,j} = \frac{|F_i \cap F_j|}{\sqrt{|F_i| \times |F_j|}}$ stands for the Cosine, $DS_{i,j} = \frac{2 \times |F_i \cap F_j|}{|F_i| + |F_j|}$ for the Dice, and $JS_{i,j} = \frac{|F_i \cap F_j|}{|F_i \cup F_j|}$ for the Jaccard similarity. Hence, $|F| = 3$.
2) *Schema-based ESDE* (**SB-ESDE**) applies the functions of SA-ESDE to every attribute of the given records. It considers every attribute independently of the others, yielding a feature vector with the Cosine, Dice and Jaccard similarity of the tokens per attribute. $|F| = 3 \times |A|$, where $|A|$ denotes the number of attributes in the input dataset.
3) *Schema-agnostic Q-gram-based ESDE* (**SAQ-ESDE**) alters SA-ESDE to work with character q-grams ($q \in [2, 10]$) instead of tokens. $F_i = [2g(r_i), 3g(r_i), ..., 10g(r_i)]$, where $xg(r_i)$ returns the set of x-grams from all attribute values of record r_i. The feature vector includes the Cosine, Dice and Jaccard similarity per q, such that $|F| = 30$.
4) *Schema-based Q-gram-based ESDE* (**SBQ-ESDE**). It alters SB-ESDE so that it works with character q-grams ($2 \leq q \leq 10$) instead of tokens. Every attribute value is con-

TABLE III
THE ESTABLISHED DATASETS FOR DL-BASED MATCHING ALGORITHMS. $|D_x|$ IS THE NUMBER OF RECORDS, $|A|$ THE NUMBER OF ATTRIBUTES, AND $|I_x|$, $|P_x|$, $|N_x|$ ARE THE NUMBERS OF LABELLED, POSITIVE AND NEGATIVE INSTANCES IN THE TRAINING (x=TR) OR THE TESTING (x=TE) SET. IR IS THE CLASS IMBALANCE RATIO.

	D_1	D_2	$\|D_1\|$	$\|D_2\|$	$\|A\|$	$\|I_{tr}\|$	$\|I_{te}\|$	$\|P_{tr}\|$	$\|P_{te}\|$	$\|N_{tr}\|$	$\|N_{te}\|$	IR	References
(a) Structured datasets													
D_{s1}	DBLP	ACM	2,616	2,294	4	7,417	2,473	1,332	444	6,085	2,029	18.0%	[10], [20], [54]
D_{s2}	DBLP	Google Scholar	2,616	64,263	4	17,223	5,742	3,207	1,070	14,016	4,672	18.6%	[10], [20], [54]
D_{s3}	iTunes	Amazon	6,907	55,923	8	321	109	78	27	243	82	24.3%	[10], [20]
D_{s4}	Walmart	Amazon	2,554	22,074	5	6,144	2,049	576	193	5,568	1,856	9.4%	[10], [19], [20], [54]
D_{s5}	BeerAdvo	RateBeer	4,345	3,000	4	268	91	40	14	228	77	14.9%	[10], [20]
D_{s6}	Amazon	Google Products	1,363	3,226	3	6,874	2,293	699	234	6,175	2,059	10.2%	[10], [19], [20], [54]
D_{s7}	Fodors	Zagat	533	331	6	567	189	66	22	501	167	11.6%	[10], [20]
(b) Dirty datasets													
D_{d1}	DBLP	ACM	2,616	2,294	4	7,417	2,473	1,332	444	6,085	2,029	18.0%	[10], [12], [20], [54]
D_{d2}	DBLP	Google Scholar	2,616	64,263	4	17,223	5,742	3,207	1,070	14,016	4,672	18.6%	[10], [12], [20], [54]
D_{d3}	iTunes	Amazon	6,907	55,923	8	321	109	78	27	243	82	24.3%	[10], [12], [20]
D_{d4}	Walmart	Amazon	2,554	22,074	5	6,144	2,049	576	193	5,568	1,856	9.4%	[10], [12], [20], [54]
(c) Textual datasets													
D_{t1}	Abt	Buy	1,081	1,092	3	5,743	1,916	616	206	5,127	1,710	10.7%	[10], [12], [19], [20]
D_{t2}	CompanyA	CompanyB	28,200	28,200	1	67,596	22,503	16,859	5,640	50,737	16,863	24.9%	[10], [20]

verted into its set of q-grams, which are then used to calculate the Cosine, Dice and Jaccard similarities of this attribute. $F_i = [2g(r_i, a_1), ..., 10g(r_i, a_{|A|})]$, where $|A|$ is the number of attributes in the input dataset. $|F| = 30\times|A|$.

5) *Schema-agnostic S-GTR-T5 ESDE* (**SAS-ESDE**). In this algorithm, the function $getFeatures()$ represents every record r_i by the 768-dimensional pre-trained Sentence-BERT (S-GTR-T5) embedding vector of the concatenation of all its attribute values, v_i. Every candidate pair $c_{i,j}$ is then represented by the feature vector $f_{i,j} = [CS_{i,j}, ES_{i,j}, WS_{i,j}]$, where $CS_{i,j}$ is the Cosine similarity of the vectors v_i and v_j, $ES_{i,j}$ is their Euclidean similarity, which is defined as $ES_{i,j}$=1/(1+$ED(v_i, v_j)$), where $ED(v_i, v_j)$ stands for the Euclidean distance of the two vectors, and $WS_{i,j}$ is the Wasserstein similarity between v_i and v_j, which is derived from the Wasserstein/Earth mover's distance [57] with the same equation combining $ES_{i,j}$ and ED. $|F| = 3$.
6) *Schema-based S-GTR-T5 ESDE* (**SBS-ESDE**) applies the previous feature generation function to each attribute. The dimensionality of its feature vector is $|F| = 3 \times |A|$, with $|A|$ denoting the number of attributes in the given dataset.

V. ANALYSIS OF EXISTING BENCHMARKS

We now assess how challenging are the datasets that are commonly used in the evaluation of DL-based matching algorithms. They were originally used to evaluate DeepMatcher and are available on its repository [58] (which provides details on their content and generation). Their characteristics are shown in Table III, where the rightmost column cites the methods from Section IV-A that used each dataset in its experiments. DeepMatcher and DITTO used all 13 datasets, while HierMatcher, EMTransformer and GNEM used 7, 5 and 3 datasets, resp. No other dataset is shared by at least two of these methods. Note that the dirty datasets, D_{d1} to D_{d4}, were generated from the structured datasets D_{s1} to D_{s4} by injecting artificial noise: for each record, the value of every attribute except "title" was randomly assigned to its "title" with 50% probability [10], [58]. Note also that each dataset is split into specific training, validation, and testing sets, with ratio 3:1:1.

A. Theoretical Measures

Degree of Linearity. The results for each benchmark dataset are shown in Figure 1. We observe that the linearity of six datasets exceeds 0.8 (with three exceeding 0.9) for both similarity measures, which indicates rather easy classification tasks, as the two classes can be separated by a linear classifier with high accuracy. The maximum values are obtained with D_{s7}, where indeed practically all DL-based algorithms achieve a perfect F1 score of $F1$=1.0 [43].

There is a wide deviation between the thresholds used by the Cosine and Jaccard similarities, but the actual difference between them is rather low: $F1^{max}_{CS}$ is higher than $F1^{max}_{JS}$ by just 0.8%, on average, across the structured and dirty datasets. In the case of textual datasets, though, Cosine similarity outperforms Jaccard similarity by 12.3% on average. This is due to the large number of tokens per record, which significantly reduces the Jaccard scores.

Overall, $F1^{max}_{CS}$ and $F1^{max}_{JS}$ suggest that *among the structured datasets, only D_{s3}, D_{s4} and D_{s6} are complex enough to call for non-linear classification models. The same applies only to D_{d3} and D_{d4} from the dirty datasets, and to both textual datasets (D_{t1} and D_{t2}).*

Complexity Measures. We now present the first complexity analysis of the main benchmark datasets for matching algorithms. All measures are implemented by the `problexity` Python package [59]. We adapted them to matching using the two-dimensional feature vector we defined at the end of Section III-B. The results appear in Figure 2.

More specifically, datasets identified as rather easy by the above linearity degree analysis achieve the lowest scores for the majority of the complexity measures, and the lowest ones on average. As expected, the lowest average score corresponds to D_{s7} (0.179), since the vast majority of its individual measures falls far below 0.2. Among the remaining easy datasets (D_{s1}, D_{s2}, D_{s5}, D_{d1} and D_{d2}), the maximum average score corresponds to D_{s5}, amounting to 0.346.

We observe that there are three datasets with an average score close to 0.346: D_{s3} with 0.354, D_{d2} with 0.341 and D_{d3}

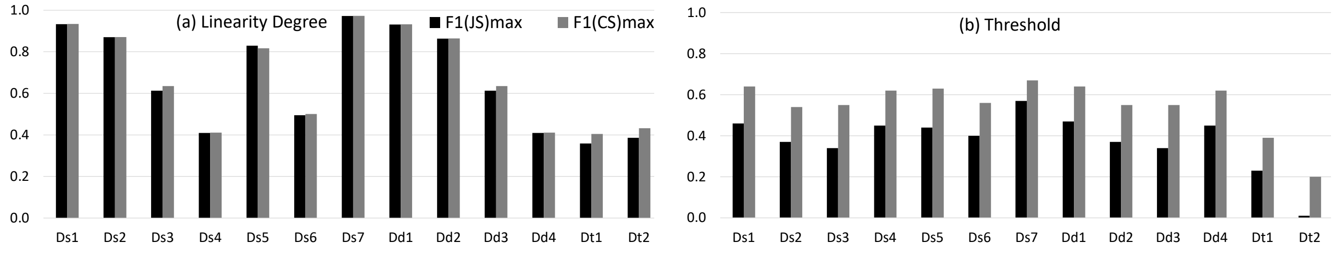

Fig. 1. Degree of linearity per dataset in Table III (left) and the respective threshold (right) with respect to $F1_{CS}^{max}$ and $F1_{JS}^{max}$ in Section III-A.

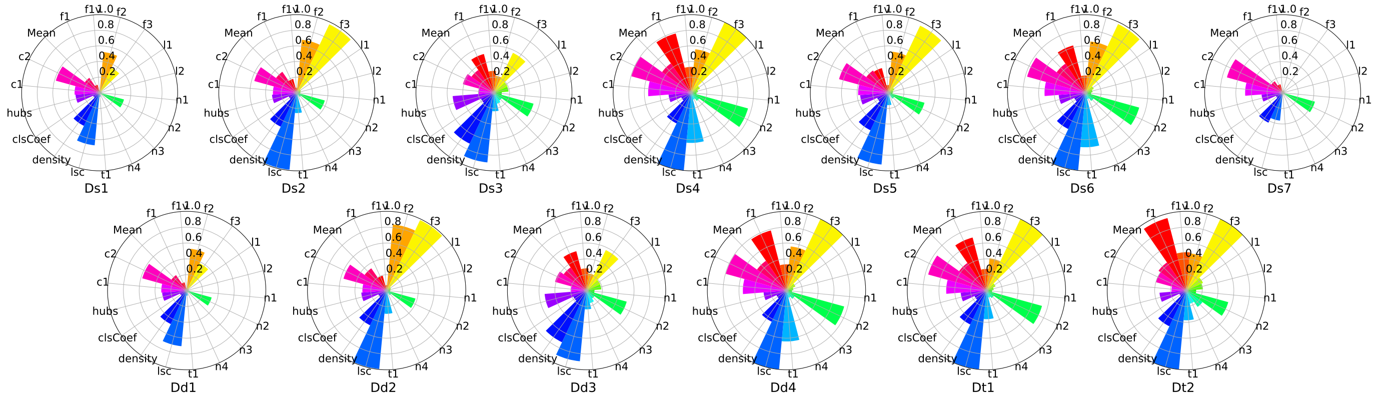

Fig. 2. Complexity measures per dataset shown in Table III. See the extension of our work in [21] for an enlarged version of these diagrams.

with 0.355. Comparing them with D_{s5}, we observe that they get higher scores for most measures. However, D_{s3} and D_{d3} achieve the minimum value for f_2 among all datasets, i.e., the overlapping region between the two classes has the smallest volume among all datasets. They also exhibit comparatively low values for f_3, which implies that the two features we have defined are quite effective in separating the two classes. Most importantly, their scores for both class imbalance measures are close to the minimum score across all datasets. The reason is that they have an unrealistically high portion of positive instances, which, as shown in Table III, is second only to that of D_{t2} (which gets the lowest scores for c_1 and c_2 in Figure 2). Regarding D_{d2}, it exhibits scores close to the minimum ones for l_1, which indicates low distances of incorrectly classified instances from a linear classification boundary, as well as for n_1 and n_2, which indicate a low number of instances surrounded by examples from the other class. As a result, *the complexity measures indicate that D_{s3}, D_{d2} and D_{d3} pose* easy classification tasks, which is in line with Table IV, where most matchers achieve high $F1$ over these datasets.

All other datasets achieve an average score that fluctuates between 0.423 and 0.457. Overall, *a mean complexity score below 0.400 indicates easy classification tasks, with only D_{s4}, D_{s6}, D_{d4}, D_{t1} and D_{t2} being challenging.*

B. Practical Measures

Setup. We conducted all experiments in Python, on a server with an Nvidia GeForce RTX 3090 GPU (24 GB RAM) and a dual AMD EPYC 7282 16-Core CPU (256 GB RAM). Given that every method depends on different Python versions and packages, we aggregated all of them into a Docker image, which facilitates the reproducibility of our experiments. Note that every evaluated method requires a different format for the input data; we performed all necessary transformations and will publish the resulting files upon acceptance (the Docker image is already public, see end of Section I).

Methods Configuration. Following [10], DeepMatcher is combined with fastText [25] embeddings in the attribute embedding module, the Hybrid model in the attribute similarity vector module, and a two layer fully connected ReLU HighwayNet [52] classifier followed by a softmax layer.

EMTransformer has two versions: EMTransformer-B and EMTransformer-R, which use BERT and RoBERTa, respectively. As noted in [60] (Section E in their Appendix), its original implementation ignores the validation set. Instead, "it reports the best $F1$ on the test set among all the training epochs". To align it with the other methods, we modified its code so that it uses the validation set to select the best performing model that is applied to the testing set.

For GNEM, we employ a BERT-based embedding model, because its dynamic nature outperforms the static pre-trained models like fastText, as shown by the authors in [25]. In the interaction module, we apply a single-layer gated graph convolution network, following the authors' recommendation.

DITTO employs RoBERTa, since it is best performing in [11], [20]. However, we were not able to run DITTO with part-of-speech tags, because these tags are provided by a service that was not available. Therefore, like all other methods we evaluated, DITTO did not employ any external knowledge.

HierMatcher employs the pre-trained fastText model [25] for embeddings, while the hidden size of each GRU layer is set to 150 in the representation layer, following [54].

We decoupled ZeroER from the blocking function that is hand-crafted for each dataset. Only in this way we can ensure that it applies to exactly the same instances as all other methods, allowing for a fair comparison.

TABLE IV
F1 per method and data set. Hyphen indicates insufficient memory. The highest F1 per category and dataset is in bold.

	D_{s1}	D_{s2}	D_{s3}	D_{s4}	D_{s5}	D_{s6}	D_{s7}	D_{d1}	D_{d2}	D_{d3}	D_{d4}	D_{t1}	D_{t2}
(a) DL-based matching algorithms													
DeepMatcher (15)	98.65	95.50	88.46	69.66	75.86	65.98	95.45	96.63	93.07	75.00	46.56	68.53	**94.04**
DeepMatcher (40)	98.76	93.70	84.62	64.42	66.67	53.73	91.67	96.54	92.73	66.67	46.99	69.21	-
DeepMatcher [10]	98.40	94.00	88.00	66.90	72.70	69.30	100.00	98.10	93.80	74.50	46.00	62.80	92.70
DITTO (15)	51.46	88.62	67.61	51.44	42.62	70.66	28.76	42.29	91.21	61.73	44.15	38.94	54.60
DITTO (40)	89.43	91.18	56.82	58.02	28.00	66.94	65.67	90.16	91.05	65.06	60.80	42.09	64.77
DITTO [20]	98.99	95.60	97.06	86.76	94.37	75.58	100.00	99.03	95.75	95.65	85.69	89.33	93.85
EMTransformer-B (15)	98.99	95.42	92.59	80.80	**82.35**	68.14	97.78	98.88	95.24	**98.04**	79.59	83.94	78.31
EMTransformer-B (40)	**99.21**	95.38	92.31	82.72	**82.35**	66.20	97.78	**98.99**	95.53	94.34	82.81	85.42	77.65
EMTransformer-R (15)	98.87	**95.90**	96.15	84.83	80.00	69.04	**100.00**	98.19	**95.78**	94.12	**83.95**	**89.29**	77.65
EMTransformer-R (40)	98.52	95.83	94.55	**85.04**	80.00	68.36	**100.00**	98.30	95.22	94.34	82.69	87.11	77.12
EMTransformer [12], [60]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	98.90	95.60	94.20	85.50	90.90	N/A
GNEM (10)	98.21	95.19	96.43	84.96	77.78	70.85	**100.00**	98.87	93.93	94.74	79.19	88.66	-
GNEM (40)	98.55	94.95	**98.18**	20.45	80.00	**74.75**	**100.00**	98.87	93.92	89.66	83.87	86.49	-
GNEM [19]	N/A	N/A	N/A	86.70	N/A	74.70	N/A	N/A	N/A	N/A	N/A	87.70	N/A
HierMatcher (10)	-	94.85	-	79.37	72.00	72.06	**100.00**	-	-	-	58.63	-	-
HierMatcher (40)	-	94.85	-	79.37	72.00	72.06	**100.00**	-	-	-	58.63	-	-
HierMatcher [54]	98.80	95.30	N/A	81.60	N/A	74.90	N/A	98.01	94.50	N/A	68.50	N/A	N/A
(b) Non-neural, non-linear ML-based matching algorithms													
Magellan-DT	97.65	86.88	88.52	62.37	84.85	54.42	**100.00**	40.07	78.76	50.00	33.89	48.46	**100.00**
Magellan-LR	97.66	88.61	84.21	65.99	80.00	44.44	**100.00**	**83.20**	76.03	50.00	32.77	37.36	**100.00**
Magellan-RF	98.32	**92.96**	**89.66**	**67.76**	**84.85**	**56.10**	**100.00**	60.47	**81.67**	**52.00**	**38.06**	**51.30**	**100.00**
Magellan-SVM	90.19	81.41	84.62	65.03	84.62	2.53	84.21	10.99	48.15	12.12	12.62	0.00	99.96
Magellan [10]	98.40	92.30	91.20	71.90	78.80	49.10	100.00	91.90	82.50	46.80	37.40	43.60	79.80
ZeroER	**98.80**	65.67	49.81	64.41	35.90	18.50	90.91	36.53	39.23	10.42	20.00	2.56	-
ZeroER [55]	96.00	86.00	N/A	N/A	N/A	48.00	100.00	N/A	N/A	N/A	N/A	52.00	N/A
(c) Non-neural, linear supervised matching algorithms													
SA-ESDE	93.06	87.57	52.94	45.27	85.71	51.58	**100.00**	92.71	86.80	52.94	**45.27**	37.67	43.97
SAQ-ESDE	93.08	**88.62**	55.81	43.91	82.76	**54.13**	97.77	93.16	**88.51**	49.41	42.82	37.94	58.40
SAS-ESDE	**93.49**	87.40	64.00	43.62	**87.50**	48.17	95.45	**93.35**	86.79	**64.00**	42.27	40.57	**79.86**
SB-ESDE	91.19	79.63	**92.31**	**67.81**	82.76	52.65	84.44	84.27	78.18	46.43	42.94	45.63	41.23
SBQ-ESDE	91.44	82.71	84.21	67.55	83.33	45.20	**100.00**	87.54	82.29	55.70	37.47	47.17	58.37
SBS-ESDE	90.89	82.45	87.72	67.35	82.76	46.68	**100.00**	85.68	80.06	43.14	41.29	**49.15**	**79.86**

Finally, we combine Magellan with four different classification algorithms: Magellan-DT uses a Decision Tree, Magellan-LR Logistic Regression, Magellan-RF a Random Forest, and Magellan-SVM a Support Vector Machine. Similar to ZeroER, we deactivated the blocking method provided by Magellan, applying it to the same blocked data sets as all other methods.

Hyperparameters. Initial experiments showed that the number of epochs is probably the most important hyperparameter for most DL-based matching algorithms. To illustrate this, we report the performance of every DL algorithm for two different settings: (i) the default number of epochs as reported in the corresponding paper, and (ii) 40 epochs, which is the most common in the original papers. Table IV shows the resulting performance, with the number in parenthesis next to each DL-based algorithm indicating the number of epochs.

Reproducibility Analysis. To verify the validity of the above configurations, which will also be applied to the new datasets, Table IV also presents the fine-tuned performance of each non-linear matching algorithm, as reported in the literature. For each method, the lower the difference between the best $F1$ performance we achieved from the one reported in the literature, the closer we are to its optimal configuration.

Our experiments with DeepMatcher exceed those of [10] in most cases, by 1.5%, on average. For EMTransformer, we consider the results reported in [60], because the original experiments in [12] show the evolution of $F1$ across the various epochs, without presenting exact numbers. The average difference with our $F1$ is just 0.15% on average. Slightly higher, albeit negligible, just 0.3%, is the mean difference between our results and GNEM's performance in [19]. Our results for HierMatcher are consistently lower than those in [54], with an average difference of 5.4%. Finally, DITTO's performance in [20] is consistently higher than our experimental results to a significant extent – on average by 25%. This is caused by the lack of external knowledge and the absence of two optimizations (see Section 3 in [20]). Yet, our best performance among all DL-based matching algorithms per dataset is very close or even higher than DITTO's performance in [20] in all datasets, but D_{s5}.

Magellan underperforms the results in [10] in D_{s4} and D_{d1}, while for D_{s1}, D_{s2}, D_{s3}, D_{d2} and D_{d4}, the differences are minor ($\leq$ 1.8% in absolute terms). For the remaining five datasets, though, our results are significantly higher than [10], by 13% on average. For ZeroER, we get slightly better performance in D_{s1}, while for the other four datasets examined in [55], our results are lower by 60%, on average. The reason is that [55] combines ZeroER with custom blocking methods and configurations in each case, whereas we use the same

TABLE V
THE NEW DATASETS GENERATED BY DEEPBLOCKER. BLOCKING PERFORMANCE IS REPORTED WITH RECALL (PC), PRECISION (PQ) AS WELL AS THE TOTAL NUMBER OF MATCHES, I.E., DUPLICATES ($|M|$), OF CANDIDATE PAIRS ($|C|$) AND OF THE MATCHING ONES ($|P|$).

	D_1	D_2	$\|D_1\|$	$\|D_2\|$	$\|M\|$	$\|A\|$	Blocking performance				DeepBlocker config.				$\|I_{tr}\|$	$\|I_{te}\|$	$\|P_{tr}\|$	$\|P_{te}\|$	$\|N_{tr}\|$	$\|N_{te}\|$	IR
							PC	PQ	$\|C\|$	$\|P\|$	$attr.$	$cl.$	K	$ind.$							
D_{n1}	Abt	Buy	1,076	1,076	1,076	3	0.899	0.029	33,356	967	name	×	31	D_2	20,014	6,671	580	193	19,433	6,478	2.9%
D_{n2}	Amazon	GP	1,354	3,039	1,104	4	0.910	0.074	13,540	1,005	title	×	10	D_1	8,124	2,708	603	201	7,521	2,507	7.4%
D_{n3}	DBLP	ACM	2,616	2,294	2,224	4	0.983	0.953	2,294	2,186	all	✓	1	D_2	1,376	459	1,312	437	65	22	95.3%
D_{n4}	IMDB	TMDB	5,118	6,056	1,968	5	0.898	0.011	158,658	1,768	all	✓	31	D_1	95,195	31,732	1,061	354	94,134	31,378	1.1%
D_{n5}	IMDB	TVDB	5,118	7,810	1,072	4	0.891	0.003	322,434	955	all	×	63	D_1	193,460	64,487	573	191	192,887	64,296	0.3%
D_{n6}	TMDB	TVDB	6,056	7,810	1,095	6	0.927	0.130	7,810	1,015	all	✓	1	D_2	4,686	1,562	609	203	4,077	1,359	13.0%
D_{n7}	Walmart	Amazon	2,554	22,074	853	6	0.894	0.018	43,418	763	all	✓	17	D_1	26,051	8,684	458	153	25,593	8,531	1.8%
D_{n8}	DBLP	GS	2,516	61,353	2,308	4	0.906	0.166	12,580	2,091	all	✓	5	D_1	7,548	2,516	1,255	418	6,293	2,098	16.6%

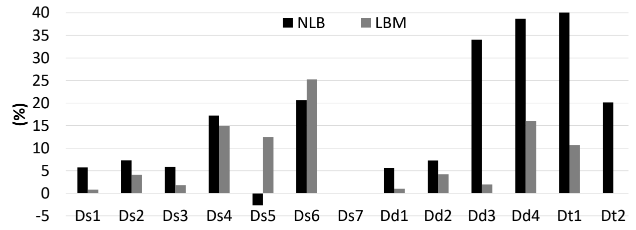

Fig. 3. Practical measures per dataset in Table III.

configuration in all datasets. Yet, the best performance per dataset that is achieved by one of Magellan's variants consistently outperforms ZeroER's performance in [55], except for D_{t1}, where its top $F1$ is 0.7% lower.

On the whole, the selected configurations provide an overall performance close to or even better than the best one for non-linear matching algorithms in the literature.

Aggregate Practical Measures. Figure 3 presents the non-linear boost (NLB) and the learning-based margin (LBM) measures per dataset. Both measures should exceed 5%, ideally 10%, in a dataset that is considered as challenging, i.e., the two classes are linearly inseparable to a large extent, leaving a significant room for improvements. *Among the structured datasets, this requirement is met only by D_{s4} and D_{s6}.* The first three datasets exhibit a high NLB, demonstrating their non-linearity, but have a very low LBM, as many algorithms achieve a practically perfect performance. In D_{s5}, there is room for improvements, but the two classes are linearly separable to a large extent, as the best linear algorithms outperform the best non-linear ones. Finally, both measures are reduced to 0 over D_{s7}, because both linear and non-linear algorithms achieve perfect $F1$. All dirty datasets have a higher degree of non-linearity, as indicated by their NLB, which consistently exceeds 5%. However, the first three are ideally solved by the DL-based matching algorithms, especially EMTransformer, *leaving D_{d4} as the only challenging dataset of this type.* Both textual datasets exhibit high non-linearly, but on D_{t2}, Magellan achieves perfect performance. *Only D_{t1} is challenging.*

Conclusion. Given that our two theoretical and two practical measures are complementary, a benchmark dataset is challenging for entity matching only if it is marked easy by none of our measures. This applies to four out of the 13 datasets, namely D_{s4}, D_{s6}, D_{d4} and D_{t1}.

VI. METHODOLOGY FOR NEW BENCHMARKS

Our methodology for creating new benchmarks consists of the following four steps:

1) Given a dataset with a complete ground truth, apply a state-of-the-art blocking method that is suitable for the data at hand. Blocking is indispensable for reducing the search space to the most likely duplicates, which can be processed by a matching algorithm within a reasonable time frame.
2) Based on the available ground truth, fine-tune the selected blocking method for a minimum level of recall. In practical situations, recall should be very high (e.g., 90%), because most learning-based matchers take decisions at the level of individual record pairs and, thus, they cannot infer duplicates not included in the candidate pairs. The fine-tuning maximizes precision for the selected recall so as to minimize the class imbalance. In this process, the selected recall level determines the difficulty of the labeled instances. The higher the recall levels are, the more difficult to classify positive instances (true matches) are included at the expense of including more and easier negative instances (true non-matches), and vice versa for low recall levels. We term "easy positive instances" those duplicates whose similarity is higher than most non-matching pairs, whereas "easy negative instances" involve non-matching entities with a similarity lower than most matching ones.
3) Randomly split the candidates pairs into training, validation and testing sets with a typical ratio, using the ground truth.
4) Apply all difficulty measures from Section III to decide whether the resulting benchmark is challenging enough.

To put this methodology into practice, we use the eight, publicly available, established datasets for RL in Table V. They cover a wide range of domains, from product matching (D_{n1}, D_{n2}, D_{n7}) to bibliographic data (D_{n3}, D_{n8}) and movies (D_{n4}-D_{n6}). We apply DeepBlocker [61] to these datasets, a generic state-of-the-art approach leveraging Autoenconder, self-supervised learning and fastText embeddings. Through grid search, DeepBlocker is configured so that its recall, also known as pair completeness (PC) [16], exceeds 90%. Note that *our methodology is generic enough to support any other blocking method and recall limit.*

For every dataset, DeepBlocker generates the candidate pair set C by indexing one of the two data sources (D_1 or D_2 in Table V), while every record of the other source is used as a query that retrieves the K most likely matches. To maximize precision, we consider the lowest K that exceeds the minimum recall. In each dataset, we use both combinations of indexing and query sets and select the one yielding the lowest number of candidates for the required recall.

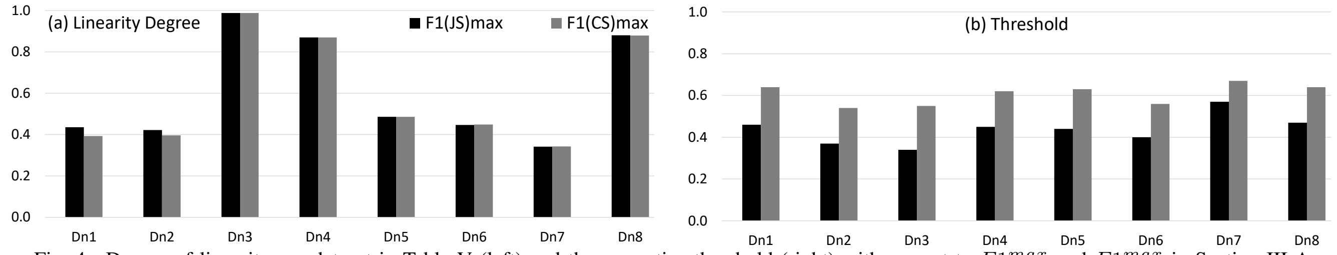

Fig. 4. Degree of linearity per dataset in Table V (left) and the respective threshold (right) with respect to $F1_{CS}^{max}$ and $F1_{JS}^{max}$ in Section III-A.

TABLE VI
F1 PER METHOD AND DATASET IN TABLE V, WITH THE BEST ONE IN BOLD. HYPHEN MEANS INSUFFICIENT MEMORY.

	D_{n1}	D_{n2}	D_{n3}	D_{n4}	D_{n5}	D_{n6}	D_{n7}	D_{n8}
(a) DL-based matching algorithms								
DM(15)	70.49	52.01	99.32	90.50	59.88	69.95	56.57	95.10
DM(40)	71.43	56.15	99.32	89.73	63.18	67.28	57.14	93.51
DITTO(15)	86.43	38.10	-	86.50	66.82	-	**71.73**	95.31
DITTO(40)	-	67.95	-	86.84	0.59	-	63.91	95.04
EM-B(15)	84.68	64.39	99.43	91.91	**67.14**	77.78	67.56	93.16
EM-B(40)	85.88	65.38	99.54	91.26	-	78.54	62.86	92.98
EM-R(15)	**91.35**	65.49	99.43	**92.51**	-	**79.28**	67.55	94.81
EM-R(40)	-	**70.12**	**99.54**	-	-	77.56	63.29	93.21
GNEM(10)	-	-	99.43	-	-	-	62.89	**95.53**
GNEM(40)	-	-	99.43	-	-	-	60.05	95.34
HM(10)	-	-	-	91.39	58.52	-	63.31	-
HM(40)	-	-	-	91.39	58.52	-	63.31	-
(b) Non-neural, non-linear ML-based matching algorithms								
MG-DT	52.55	41.67	99.54	91.69	59.72	56.84	50.00	91.73
MG-LR	43.84	39.19	99.66	91.25	59.64	**61.10**	55.65	91.06
MG-RF	**57.42**	**44.44**	**99.66**	**92.64**	**61.11**	59.74	61.18	**93.82**
MG-SVM	-	-	98.20	91.01	59.34	61.01	**61.67**	88.70
ZeroER	32.66	22.14	99.32	43.32	0.50	53.76	61.52	84.14
(c) Non-neural, linear supervised matching algorithms								
SA-ESDE	47.79	40.35	98.64	**85.75**	47.86	43.98	34.41	88.24
SAQ-ESDE	44.59	41.41	98.64	82.80	49.93	43.96	37.77	88.57
SAS-ESDE	47.97	39.58	98.75	77.41	49.53	44.22	35.19	87.47
SB-ESDE	49.62	46.87	**99.66**	61.95	**58.87**	**60.50**	**66.13**	89.95
SBQ-ESDE	52.95	**49.79**	**99.66**	20.00	7.61	54.26	34.07	**91.36**
SBS-ESDE	**53.65**	45.39	**99.66**	20.00	7.61	53.60	33.43	88.29

We also fine-tune two more hyperparameters: (1) whether cleaning is used or not (if it does, stop-words are removed and stemming is applied to all words), and (2) the attributes providing the values to be blocked. We consider all individual attributes as well as a schema-agnostic setting that concatenates all attributes into a sentence. DeepBlocker uses fastText to convert these attributes into embedding vectors. fastText's static nature ensures that the order of words in the concatenated text does not affect the resulting vector. For every hyperparameter, we consider all possible options and select the one minimizing the returned set of candidates, i.e., maximizing precision, also known as pairs quality (PQ) [16].

The exact configuration of DeepBlocker per dataset is shown in Table V. Column *attr.* indicates that the schema-agnostic setting yields the best performance in most cases, column *cl.* suggests that cleaning is typically required, and column *ind.* shows that the smallest data source is typically indexed. Finally, the number of candidates per query entity, K, differs widely among the datasets.

Given that DeepBlocker constitutes a stochastic approach, the performance reported in Table V corresponds to the average after 10 repetitions. For this reason, in some cases, PC drops slightly lower than 0.9. The resulting candidate pairs are randomly split into training, validation, and testing sets with the same ratio as the benchmarks in Table III (3:1:1). These settings simulate the realistic scenarios, where blocking is applied to exclude obvious non-matches, and then a subset of the generated candidate pairs is labelled to train a matching algorithm that resolves the rest of the candidates. The instances per class and set are reported in Table V (note that the testing and validation sets have the same size, while the imbalance ratio in the rightmost column is the same in all sets).

TABLE VII
EXISTING VS NEW BENCHMARKS

	PC	PQ	IR		PC	PQ	IR
D_{t1}	0.955	0.120	12.03%	D_{n1}	0.899	0.029	2.90%
D_{s1}	0.998	0.137	13.68%	D_{n3}	0.983	0.953	95.30%
D_{s2}	1.000	0.229	22.89%	D_{n8}	0.906	0.166	16.60%
D_{s4}	1.000	0.104	10.37%	D_{n7}	0.894	0.018	1.80%
D_{s6}	0.898	0.247	24.66%	D_{n2}	0.910	0.074	7.40%

At this point, it is worth juxtaposing the existing and the new benchmarks that have the same origin. These datasets are compared in Table VII. We observe that D_{t1} and D_{s2} outperform D_{n1} and D_{n8}, respectively, both in terms of recall and precision, even though DeepBlocker outperforms Magellan's blocking methods [61]. Therefore, the higher precision of D_{t1} and D_{s2} is most likely achieved due to the removal of negative pairs. Moreover, the recall of D_{s4} is lower than D_{s7} by just 6%, but its precision is higher by a whole order of magnitude, whereas the recall of D_{s6} is lower than D_{n2} by just 1.2%, but its precision is higher by 3.3 times. These tradeoffs are not common in blocking over these two particular datasets [62] and could be caused by removing a large portion of negative pairs. Finally, D_{n3} exhibits much higher precision (almost by 7 times) than D_{s1}, even though their difference in recall is just 1.5%. Given that a wide range of blocking methods achieves exceptionally high precision in this bibliographic dataset [62], the low precision of the existing benchmark could be caused by including a large number of easy, negative pairs, i.e. obvious non-matches. Overall, *the five existing benchmarks in Table VII seem to involve an undocumented approach for inserting or removing an arbitrary number of negative pairs.*

A. Analysis of New Benchmarks

The above process is not guaranteed to yield challenging classification tasks for learning-based matching algorithms. For this reason, we include a third step, which assesses the difficulty of each new benchmark through the theoretical and practical measures defined in Sections III and IV, respectively.

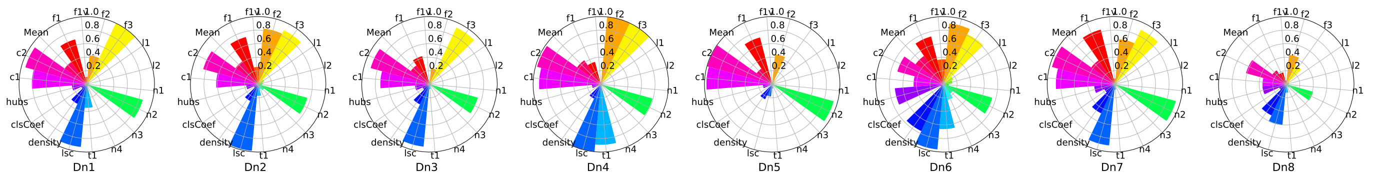

Fig. 5. Complexity measures per dataset in Table V. See the extension of our work in [21] for an enlarged version of these diagrams.

1) Theoretical Measures: **Degree of Linearity.** In Figure 4(a), we show the values of $F1^{max}_{CS}$ and $F1^{max}_{JS}$ per dataset along with the respective thresholds. We observe that both measures exceed 0.87 for D_{n3}, D_{n4} and D_{n8}, while remaining below 0.49 for all other datasets. For D_{n3}, this is expected, because, as explained above, it involves quite unambiguous duplicates, due to the low levels of noise. The same is also true for D_{n8}, which also conveys bibliographic data, with its duplicates sharing clean and distinguishing information. The opposite is true for D_{n4}: its low precision after blocking indicates high levels of noise and missing values, requiring many candidates per entity to achieve high recall. The fastText embeddings may add to this noise, as the attribute values are dominated by movie titles and the names of actors and directors, which are underrepresented in its training corpora. In contrast, the traditional textual similarity measures at the core of $F1^{max}_{CS}$ and $F1^{max}_{JS}$ are capable of separating linearly the two matching classes.

On the whole, these a-priori measures suggest that D_{n1}, *D_{n2}, D_{n5}, D_{n6} and D_{n7} pose challenging matching tasks.*

Complexity Measures. These are presented in Figure 5. The average complexity score is lower than 0.40 for D_{n3} and D_{n8} (0.339 and 0.251, respectively), in line with the degree of linearity, but it exceeds this threshold for D_{n4} (0.431). This is caused by its very low imbalance ratio (see also Table V), which results in high scores for the class imbalance measures and some of the feature overlapping ones. In all other cases, though, it exhibits the (second) lowest score among all datasets, including the established ones. Note also that D_{n5} yields a very low average score (0.282), that surpasses only D_{n8}. This indicates a rather easy classification task, because of the very low values ($\ll$0.2) for 9 out of the 17 complexity measures. Hence, only *D_{n1}, D_{n2}, D_{n6} and D_{n7} correspond to challenging, non-linearly separable matching benchmarks.*

2) Practical Measures: For the DL-based matching algorithms, we use the same configurations as for the existing benchmarks in Table IV, due to their high performance, which matches or surpasses the literature. The results appear in Table VI, while Figure 5 reports the corresponding non-linear boost (NLB) and the learning-based margin (LBM).

For the datasets marked as challenging by all theoretical measures (D_{n1}, D_{n2}, D_{n6} and D_{n7}), both practical measures take values well above 5%. LBM takes its minimum value (8.7%) over D_{n1}, as EMTransformer with RoBERTa performs exceptionally well, outperforming all other DL-based algorithms by at least 5% and all others by at least 34%. NLB takes its minimum value over D_{n7}, because the $F1$ for SB-ESDE is double as that of all other linear algorithms, reducing is distance from the top DL-based one to 5.6%.

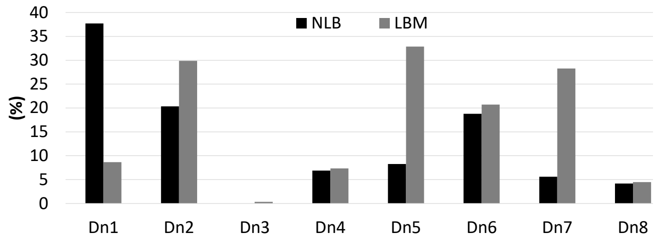

Fig. 6. Practical measures per dataset in Table V.

All algorithms achieve perfect performance over D_{n3}, thus reducing both practical measures to 0. The same applies to a lesser extent to D_{n8}, where both measures amount to around 4.3%. In D_{n4} and D_{n5}, both practical measures exceed 5% to a significant extent. The reason for the former is that the best DL- and ML-matchers lie in the middle between the perfect $F1$ and the best linear algorithm, whose performance matches the degree of linearity. For D_{n5}, the practical measures are in line with the degree of linearity, unlike the complexity measures, which suggest low levels of difficulty.

Overall, the practical measures suggest that, with the exception of D_{n3} and D_{n8} (which exhibit linear separability of their classes), *all other datasets are challenging enough for assessing the relative performance of DL-based matchers*. This means that D_{n1}, D_{n2}, D_{n6} and D_{n7} are marked as challenging by all theoretical and practical measures.

VII. Conclusions

We make the following observations: (1) The datasets used for benchmarking ER matching algorithms should be evaluated both a-priori, through their degree of linearity and complexity, and a-posteriori, through the aggregate measures summarizing the performance of linear and non-linear matchers. Excelling in all these respects is necessary for datasets that leave enough room for improvements by complex, learning-based classifiers. (2) Most of the popular datasets used as benchmarks for DL-based matchers involve almost linearly separable candidate pairs, or are perfectly solved by most existing matching algorithms (therefore, leaving no room for improvement). These characteristics render these datasets *unsuitable* for benchmarking matching algorithms. (3) We experimentally demonstrate that our proposed methodology for creating new ER benchmarks leads to datasets that are better suited for assessing the benefits of DL-based matchers.

In the future, we plan to examine whether different configurations can extract challenging benchmarks from one of the easy datasets, like the bibliographic ones (e.g., DBLP-ACM). We actually intend to create a series of datasets that cover the entire continuum of benchmark difficulty.

Acknowledgments. This research was partially funded by the Horizon Europe project STELAR (GA No. 101070122).

References

[1] F. Naumann and M. Herschel, *An Introduction to Duplicate Detection*, ser. Synthesis Lectures on Data Management. Morgan and Claypool Publishers, 2010.

[2] H. Newcombe, J. Kennedy, S. Axford, and A. James, "Automatic linkage of vital records," *Science*, vol. 130, no. 3381, pp. 954–959, 1959.

[3] O. Binette and R. C. Steorts, "(Almost) all of entity resolution," *Science Advances*, vol. 8, no. 12, p. eabi8021, 2022.

[4] P. Christen, T. Ranbaduge, and R. Schnell, *Linking Sensitive Data – Methods and Techniques for Practical Privacy-Preserving Information Sharing*. Heidelberg: Springer, 2020.

[5] X. L. Dong and D. Srivastava, *Big Data Integration*, ser. Synthesis Lectures on Data Management. Morgan and Claypool Publishers, 2015.

[6] G. Papadakis, E. Ioannou, E. Thanos, and T. Palpanas, *The Four Generations of Entity Resolution*, ser. Synthesis Lectures on Data Management. Morgan and Claypool, 2021.

[7] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas, "Blocking and filtering techniques for entity resolution: A survey," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–42, 2020.

[8] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra, "Magellan: Toward building entity matching management systems," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1197–1208, 2016.

[9] N. Barlaug and J. A. Gulla, "Neural networks for entity matching: A survey," *Transactions on Knowledge Discovery from Data*, vol. 15, no. 3, 2021.

[10] 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*, 2018, pp. 19–34.

[11] Y. Li, J. Li, Y. Suhara, J. Wang, W. Hirota, and W. Tan, "Deep entity matching: Challenges and opportunities," *ACM J. Data Inf. Qual.*, vol. 13, no. 1, pp. 1:1–1:17, 2021.

[12] U. Brunner and K. Stockinger, "Entity matching with transformer architectures - A step forward in data integration," in *EDBT*, 2020, pp. 463–473.

[13] T. Wang, H. Lin, C. Fu, X. Han, L. Sun, F. Xiong, H. Chen, M. Lu, and X. Zhu, "Bridging the gap between reality and ideality of entity matching: A revisiting and benchmark re-construction," *arXiv preprint arXiv:2205.05889*, 2022.

[14] V. Christophides, V. Efthymiou, and K. Stefanidis, *Entity Resolution in the Web of Data*, ser. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2015.

[15] P. Christen, D. J. Hand, and N. Kirielle, "A review of the f-measure: Its history, properties, criticism, and alternatives," *ACM Comput. Surv.*, 2023.

[16] P. Christen, *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, ser. Data-Centric Systems and Applications. Springer, 2012.

[17] D. J. Hand and P. Christen, "A note on using the F-measure for evaluating record linkage algorithms," *Statistics and Computing*, vol. 28, no. 3, pp. 539–547, 2018.

[18] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "An overview of end-to-end entity resolution for big data," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 127:1–127:42, 2021.

[19] R. Chen, Y. Shen, and D. Zhang, "GNEM: A generic one-to-set neural entity matching framework," in *WWW*, 2020, pp. 1686–1694.

[20] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, "Deep entity matching with pre-trained language models," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 50–60, 2020.

[21] G. Papadakis, N. Kirielle, P. Christen, and T. Palpanas, "A critical re-evaluation of benchmark datasets for (deep) learning-based matching algorithms," *arXiv preprint arXiv: 2307.01231*, 2023.

[22] V. H. Barella, L. P. F. Garcia, M. C. P. de Souto, A. C. Lorena, and A. C. P. L. F. de Carvalho, "Data complexity measures for imbalanced classification tasks," in *IEEE International Joint Conference on Neural Networks IJCNN*, 2018, pp. 1–8.

[23] A. Lorena, L. Garcia, J. Lehmann, M. Souto, and T. Ho, "How complex is your classification problem," *A survey on measuring classification complexity. arXiv*, 2018.

[24] J. C. Gower, "A general coefficient of similarity and some of its properties," *Biometrics*, pp. 857–871, 1971.

[25] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[28] 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*. San Francisco, USA: ACM, 2019, pp. 2413–2424.

[29] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, "Low-resource deep entity resolution with transfer and active learning," in *Annual Meeting of the Association for Computational Linguistics*, Florence, 2019, pp. 5851–5861.

[30] R. Peeters and C. Bizer, "Dual-objective fine-tuning of BERT for entity matching," *Proc. VLDB Endow.*, vol. 14, no. 10, pp. 1913–1921, 2021.

[31] R. Zhang, B. D. Trisedya, M. Li, Y. Jiang, and J. Qi, "A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning," *The VLDB Journal*, pp. 1–26, 2022.

[32] M. Leone, S. Huber, A. Arora, A. García-Durán, and R. West, "A critical re-evaluation of neural methods for entity alignment," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1712–1725, 2022.

[33] 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 *International Conference on Information and Knowledge Management*. Beijing, China: ACM, 2019, pp. 629–638.

[34] B. Li, W. Wang, Y. Sun, L. Zhang, M. A. Ali, and Y. Wang, "Grapher: Token-centric entity resolution with graph convolutional neural networks," in *AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8172–8179.

[35] Z. Wang, B. Sisman, H. Wei, X. L. Dong, and S. Ji, "Cordel: A contrastive deep learning approach for entity linkage," in *ICDM*, 2020, pp. 1322–1327.

[36] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *SIGMOD*, 2020, pp. 1335–1349.

[37] Z. Zhao and R. C. Fernandez, "Leva: Boosting machine learning performance with relational embedding data augmentation," in *SIGMOD*, 2022, pp. 1504–1517.

[38] A. Jain, S. Sarawagi, and P. Sen, "Deep indexed active learning for matching heterogeneous entity representations," *Proceedings of the VLDB Endowment*, vol. 15, no. 1, pp. 31–45, 2021.

[39] J. Tu, J. Fan, N. Tang, P. Wang, C. Chai, G. Li, R. Fan, and X. Du, "Domain adaptation for deep entity resolution," in *ACM Conference on Management of Data*, Philadelphia, 2022.

[40] J. Foxcroft, A. d'Alessandro, and L. Antonie, "Name2vec: Personal names embeddings," in *Canadian Conference on Artificial Intelligence*. Springer, 2019, pp. 505–510.

[41] M. Paganelli, F. Del Buono, P. Marco, F. Guerra, and M. Vincini, "Automated machine learning for entity matching tasks," in *EDBT*, 2021.

[42] K. Zeng, C. Li, L. Hou, J. Li, and L. Feng, "A comprehensive survey of entity alignment for knowledge graphs," *AI Open*, vol. 2, pp. 1–13, 2021.

[43] D. Zhang, Y. Nie, S. Wu, Y. Shen, and K. Tan, "Multi-context attention for entity matching," in *WWW*, 2020, pp. 2634–2640.

[44] Z. Yao, C. Li, T. Dong, X. Lv, J. Yu, L. Hou, J. Li, Y. Zhang, and Z. Dai, "Interpretable and low-resource entity matching via decoupling feature learning from decision making," in *ACL/IJCNLP*, 2021, pp. 2770–2781.

[45] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1454–1467, 2018.

[46] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[48] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[49] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, “Albert: A lite bert for self-supervised learning of language representations,” *arXiv preprint arXiv:1909.11942*, 2019.

[50] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, “Transformer models for text-based emotion detection: a review of bert-based approaches,” *Artif. Intell. Rev.*, vol. 54, no. 8, pp. 5789–5829, 2021.

[51] M. Paganelli, F. D. Buono, A. Baraldi, and F. Guerra, “Analyzing how BERT performs entity matching,” *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1726–1738, 2022.

[52] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, “Recurrent highway networks,” in *ICML*, vol. 70, 2017, pp. 4189–4198.

[53] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *NAACL-HLT*, 2019, pp. 4171–4186.

[54] C. Fu, X. Han, J. He, and L. Sun, “Hierarchical matching network for heterogeneous entity resolution,” in *IJCAI*, 2020, pp. 3665–3671.

[55] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, “Zeroer: Entity resolution using zero labeled examples,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1149–1164.

[56] T. Herzog, F. Scheuren, and W. Winkler, *Data Quality and Record Linkage Techniques*. Springer Verlag, 2007.

[57] C. Villani and C. Villani, “The wasserstein distances,” *Optimal Transport: Old and New*, pp. 93–111, 2009.

[58] “Deepmatcher data repository,” 2018, https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md.

[59] J. Komorniczak, , and P. Ksieniewicz, “problexity — an open-source python library for binary classification problem complexity assessment,” *arXiv preprint arXiv:2207.06709*, 2022.

[60] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, “Deep entity matching with pre-trained language models,” *CoRR*, vol. abs/2004.00584, 2020.

[61] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. M. Fung, and A. Doan, “Deep learning for blocking in entity matching: A design space exploration,” *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2459–2472, 2021.

[62] G. Papadakis, M. Fisichella, F. Schoger, G. Mandilaras, N. Augsten, and W. Nejdl, “How to reduce the search space of entity resolution: with blocking or nearest neighbor search?” *CoRR*, vol. abs/2202.12521, 2022. [Online]. Available: https://arxiv.org/abs/2202.12521