





# Deduplication Over Heterogeneous Attribute Types (D-HAT)

Loujain Liekah<sup>1</sup>(✉)  and George Papadakis<sup>2</sup> 

<sup>1</sup> University of Claude Bernard Lyon 1, Villeurbanne, France  
loujain.liekah5@gmail.com

<sup>2</sup> National and Kapodistrian University of Athens, Athens, Greece  
gpapadis@di.uoa.gr

**Abstract.** Deduplication is the task of recognizing multiple representations of the same real-world object. The majority of existing solutions focuses on textual data, this means that data sets containing boolean and numerical attribute types are rarely considered in the literature, while the problem of missing values is inadequately covered. Supervised solutions cannot be applied without an adequate number of labelled examples, but training data for deduplication can only be obtained through time-costly processes. In high dimensional data sets, feature engineering is also required to avoid the risk of overfitting. To address these challenges, we go beyond existing works through D-HAT, a clustering-based pipeline that is inherently capable of handling high dimensional, sparse and heterogeneous attribute types. At its core lies: (i) a novel matching function that effectively summarizes multiple matching signals, and (ii) *MutMax*, a greedy clustering algorithm that designates as duplicates the pairs with a mutually maximum matching score. We evaluate D-HAT on five established, real-world benchmark data sets, demonstrating that our approach outperforms the state-of-the-art supervised and unsupervised deduplication algorithms to a significant extent.

**Keywords:** Clustering · Entity matching · Data quality

## 1 Introduction

Integrating overlapping and complementary data sets is a common process that creates new and valuable knowledge [3]. The main task of integration is to identify *duplicate records*, which represent the same real-world entity, such as products, institutes, or patients. This task is called *deduplication* [8], entity matching [13], entity resolution [19] or record linkage [10]. It constitutes a crucial task that improves the data quality by repairing and curating data sources [9], reducing the storage size, and preparing data for downstream applications [8].

Existing solutions for deduplication are based on calculating pairwise similarity scores from one or more attributes [6]. The *unsupervised* methods create

---

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 875171.

a similarity graph, where the nodes correspond to records and the edges are weighted by the matching scores of the adjacent nodes [12]. The graph is then partitioned into clusters such that all nodes within each cluster correspond to duplicate records. These approaches typically calculate matching scores by treating all attributes as textual data [6]. However, real-world data sets involve heterogeneous attribute types, i.e., numerical, categorical and boolean attributes. Casting these types as strings disregards important information and possibly leads to inaccurate matching scores. For example, the prices “14” and “14.00” are identical as numbers, but partially similar when compared as sequences of characters and totally dissimilar when treated as tokens. Hence, unsupervised techniques need to correctly model and support heterogeneous attribute types.

On the other hand, *supervised* methods typically model deduplication as a binary classification task [13]. They convert each pair of records into a feature vector by applying similarity metrics on different attributes. The vectors are then labelled to train a classifier that predicts the matching status for unlabelled pairs. However, these approaches face multiple challenges: (i) The curse of dimensionality, i.e., tasks become exceedingly difficult with a higher number of dimensions. (ii) Labeled data is scarce, but obtaining it through crowd-sourcing is costly and time-consuming [22]. Moreover, its size and quality affects the end result to a significant extent [17], but are hard to ensure, due to the heavy class imbalance. (iii) Supervised methods require long training times [17].

To address these shortcomings, we introduce D-HAT (Deduplication with Heterogeneous Attribute Types), a novel clustering-based pipeline for end-to-end deduplication. D-HAT goes beyond existing works in three ways: (i) It inherently supports data sets with heterogeneous types of attributes and a large portion of missing values (i.e., high sparsity). (ii) It inherently supports and leverages complex schemata of high dimensionality. (iii) It achieves state-of-the-art results without requiring any labelled data. Our contributions are the following:

- We propose D-HAT, an automated end-to-end, clustering-based framework for deduplicating high-dimensional data sets with heterogeneous attribute types and missing values. Its matching algorithm uses as features a comprehensive set of signals, coupling them with a novel greedy clustering method that defines as matches the records with mutually maximum matching scores.
- We conduct experiments on established real benchmark data sets, showing that: (i) In terms of effectiveness, D-HAT outperforms the state-of-the-art supervised and unsupervised baseline methods. (ii) In terms of time efficiency, D-HAT has an undeniable advantage over the baseline methods.
- We have publicly released all data and code used in our experiments through <https://github.com/Loujainl/D-HAT>.

## 2 Related Work

The growing research on deduplication reflects its increasing importance, with numerous methods tackling various aspects [4, 6, 8].

One of deduplication’s main challenges is its quadratic complexity: in the worst case, it examines all possible pairs of records. *Blocking* is typically used

to alleviate this complexity and to scale deduplication to voluminous data sets [5, 19]. Blocking puts together similar records in groups called blocks by applying blocking schemes or functions. A blocking function extracts signatures from every record, dividing the input data set into a set of overlapping blocks – comparisons are reduced to *candidates*, i.e., pairs of records sharing at least one block, reducing the computational cost to a significant extent. Yet, the higher time efficiency comes with the risk of missing potential matches [20].

After blocking, *matching* is performed to determine the degree of similarity between the candidate pairs of records. In essence, it applies similarity functions to the values of selected attributes of the candidate records, obtaining numerical matching scores. Next, it determines whether the resulting degree of similarity is sufficient for designating two records as duplicates. We distinguish the matching algorithms into unsupervised and supervised ones.

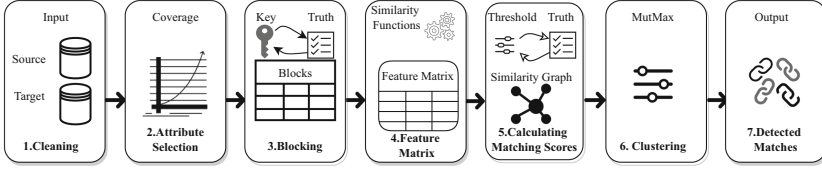
The former category includes a collection of methods that are provided by JedAI [18, 21] and Stringer [12], with *ZeroER* [24] constituting the state-of-the-art unsupervised approach; it represents every candidate pair as a feature vector. Unlike supervised methods, it does not require a training set. Instead, at its core lies the observation that the distribution of the feature vectors for duplicate records differs from that of the non-matching records. Based on this idea, it learns the parameters of the Gaussian distribution of matching vectors by iteratively applying expectation maximization to compute the posterior probability of a matching label given the feature vector. A posterior probability higher than 0.5 is considered as an indication of duplicate records.

Among the supervised methods, the most popular one is *Magellan* [13], a system that combines a variety of features with the main machine learning classifiers, such as decision trees, logistic regression and support vector machines. After providing an annotated sample of candidate pairs  $T$ , matching is performed by training a classifier over  $T$ . *Magellan* also offers a set of blocking methods.

DeepMatcher [17] is a space of matching solutions based on neural networks with three modules: i) attribute embedding, ii) attribute similarity representation, and iii) a classification module. In most cases, the first module relies on pre-trained fastText embeddings [1] to convert every token to a vector. EMTransformer [2] and DITTO [15] go beyond DeepMatcher by leveraging attention-based transformers like BERT [7], and RoBERTa [16]. These solutions perform well on textual data, outperforming *Magellan* in terms of accuracy [2, 15, 17]. We disregard them, as they require large training sets and many hours of training [17] in order to fine-tune hundreds of thousands of parameters [23].

### 3 Preliminaries

A *data set*  $T$  is a collection of records. A *record* is an object description denoted by  $r_i$ , where  $i$  is a unique identifier. Records are defined by their *attributes*. The set of attributes in  $T$  is denoted by  $T.A$ , while the value of a specific attribute  $a$  in record  $r_i$  is symbolized as  $r_i.a$ ;  $r_i.a = N/A$  indicates that  $r_i$  lacks a value for  $a$ , i.e., there is a missing or a null value. Two records,  $r_i$  and  $r_j$ , that describe the same real-world object are *matching*, i.e., *duplicates*, a situation denoted by  $r_i \equiv r_j$ . A data set is called *clean* if it does not contain any duplicates.



**Fig. 1.** The end-to-end pipeline of D-HAT.

*Deduplication* is the task of identifying and linking duplicate records. A characteristic of this task is that the number of duplicate records scales linearly with the size of the input, unlike its computational cost, which increases quadratically [11]. As a result, Deduplication constitutes a heavily imbalanced task and its effectiveness is measured with respect to the following measures:

1. *Recall*, the portion of existing duplicates that are detected, i.e.,  $Re = \frac{TP}{TP+FN}$ .
2. *Precision*, the portion of record pairs characterized as duplicates that are indeed matching, i.e.,  $Pr = \frac{TP}{TP+FP}$ .
3. *F-Measure*, the harmonic mean of Recall and Precision,  $F1 = 2 \times \frac{Pr \times Re}{Pr + Re}$ ,

where TP stands for the true positive pairs, FP for the false positive ones, and FN for the false negative ones.

In this context, Deduplication can be formally defined as follows:

*Problem 1 (Deduplication).* Given a data set  $T$ , detect the set of duplicate pairs of records,  $D = \{r_i, r_j \in T : i \neq j \wedge r_i \equiv r_j\}$ , such that Recall, Precision and F-Measure are maximized.

## 4 Our Approach

We now delve into our framework, whose pipeline is illustrated in Fig. 1.

**Step 1: Data Cleaning.** The first step prepares the input by determining the core characteristics of the attributes describing the given data set(s)<sup>1</sup>, i.e., it calculates the number of unique values and the data type per attribute. Attributes that have two unique values are converted to boolean to obtain a more precise degree of similarity. Attributes with very few unique values (<10) are treated as categorical variables. Numerical attributes are identified through regular expressions that detect quantities, possibly accompanied by an optional unit of measurement. E.g., an attribute value `width = "42.8 in"` is transformed into `width = 42.8` and is marked as a numeric data type. Min-max normalization is then performed on the values of numeric attributes:

**Step 2: Attribute Selection.** Attributes with a majority of missing values lack valuable information for deduplication and, thus, can be disregarded. The *coverage of an attribute  $a$*  expresses the portion of non-empty values in  $a$  across

<sup>1</sup> In the case of Record Linkage, we assume aligned schemata.

all input records; the fewer missing values there are, the higher is the coverage. We formally define the coverage  $c$  of each attribute as:  $c(a) = 1 - \frac{|r_i.a=N/A:r_i \in T|}{|T|}$ .

This step discards the attributes with a coverage below a specific threshold. Preliminary experiments demonstrated that 0.1 constitutes an effective value.

**Step 3: Blocking.** This step is critical because it determines two things:

1. Time efficiency, because the processing time of the following steps is determined by the number of candidates in the resulting blocks.
2. Effectiveness, because the recall of D-HAT is bounded by the recall of blocking; the false negative pairs of records, which have no block in common, cannot be detected by the subsequent steps, and are excluded from the final output.

Therefore, it is crucial that blocking balances these two competing goals: the reduced search space and the high effectiveness. D-HAT is generic enough to accommodate any blocking method that meets this requirement. Preliminary experiments indicated that Magellan’s [13] *overlap blocker* is a robust approach for creating blocks of high performance (see Sect. 5 for more details). It defines as candidate pairs those sharing at least one token in the values of a specific attribute. D-HAT applies the overlap blocker to all textual attributes in the given data sets and opts for the one minimizing the number of candidates, while maximizing coverage – high coverage implicitly signals high recall after blocking.

**Step 4: Feature Matrix.** Similar to supervised approaches, D-HAT represents each pair of records as a feature vector by applying type-specific normalized similarity functions to selected attributes. Unlike supervised approaches, these vectors are unlabelled. In more detail, after detecting the type of every attribute in Step 1, D-HAT creates a feature vector  $V_{i,j}$  for each candidate pair of records  $(r_i, r_j) \in B$ , where  $B$  is the set of blocks produced by the previous step and the  $k^{th}$  feature/dimension in  $V_{i,j}$ ,  $V_{i,j}^k$ , stems from a similarity function that is compatible with the type of the  $k^{th}$  attribute,  $a_k$ . If the value of either record for  $a_k$  is empty or incorrect (i.e., incompatible with the type of  $a_k$ ),  $V_{i,j}^k = \text{'N/A'}$ , which stands for a missing feature. Note that this step does not require any domain knowledge from the user. D-HAT automatically detects the attribute type and applies the appropriate similarity functions in order to create the features.

In particular, the following functions are used by D-HAT:

- For boolean and categorical attributes, the equality operator.
- For numerical attributes, four similarity functions are used:
  1. The equality operator,
  2. The Euclidean similarity,  $V_{i,j}^k = 1 - \text{EuclidDist}(r_i.a_k, r_j.a_k)$ .
  3. The relative similarity,  $V_{i,j}^k = 1 - \frac{|r_i.a_k - r_j.a_k|}{\max(r_i.a_k, r_j.a_k)}$ .
  4. The normalized Manhattan similarity,  $V_{i,j}^k = \frac{|r_i.a_k - r_j.a_k|}{\max(r_i.a_k, r_j.a_k)}$ .
- For textual attributes, the following functions are used:
  - (i) *Syntactic similarity measures.*

D-HAT distinguishes textual attributes into short strings, if their average value entails less than five words, and long strings otherwise. For both types, it employs the following functions:

1. Jaccard similarity:  $V_{i,j}^k = \frac{|token\_set(r_i.a_k) \cap token\_set(r_j.a_k)|}{|token\_set(r_i.a_k) \cup token\_set(r_j.a_k)|}$ .
2. Generalized Jaccard, which extends the previous measure to consider the bags of tokens:  $V_{i,j}^k = \frac{|bag(r_i.a_k) \cap bag(r_j.a_k)|}{|bag(r_i.a_k) \cup bag(r_j.a_k)|}$ .
3. Overlap Coefficient:  $V_{i,j}^k = \frac{|token\_set(r_i.a_k) \cap token\_set(r_j.a_k)|}{\min(|token\_set(r_i.a_k)|, |token\_set(r_j.a_k)|)}$ .
4. Bag:  $V_{i,j}^k = 1 - \frac{\max(|bag(r_i.a_k) - bag(r_j.a_k)|, |bag(r_j.a_k) - bag(r_i.a_k)|)}{\max(|(r_j.a_k)|, |(r_i.a_k)|)}$ .
5. Dice Similarity:  $V_{i,j}^k = 2 \times \frac{|token\_set(r_i.a_k) \cap token\_set(r_j.a_k)|}{|token\_set(r_i.a_k)| + |token\_set(r_j.a_k)|}$ .

Additionally, D-HAT uses two similarity functions for short strings:

- Levenshtein similarity, the minimum number of edit operations (insert, delete or substitute) required to transform one string to another.
  - Hamming, similar to Levenshtein except that it allows only substitution.
- (ii) *Semantic similarity measures.* D-HAT exploits pre-trained embedding representations of textual data. Two types of representations are actually used:
- a) *Word-based models* like word2vec and GlobalVectors (GloVe). They substitute each token (word) by a meaningful numeric vector that is learnt from training a shallow feedforward neural network on large, external, un-annotated textual corpora, such as Google News and Wikipedia. In these models, words with contextual similarity have linearly related vector representations. However, they cannot produce vector representations for words that are *out-of-vocabulary*.
  - b) To address this limitation, *skipgram models* like fastText [1] represent each word by the sum of the vector representations of its bag of characters. Thus, they are capable of learning a recurrent neural network that yields vector representations for words, independently of their occurrence in the training data.

To extract numeric features/dimensions from the three pre-trained embeddings (i.e., word2vec, GloVe and fastText), D-HAT applies three similarity functions to the vectors of two records: the cosine, the Euclidean and the word mover's similarity [14]. For the last two functions, the homonymous distance function  $d$  is transformed into a similarity value  $sim$  as follows:  $sim = \frac{1}{1+d}$ .

- (iii) *Hybrid similarity measures.* This configuration combines the aforementioned syntactic similarity measures with the semantic ones, given that they capture complementary matching evidence.

Overall, D-HAT creates one feature per boolean and categorical attributes, four per numeric ones as well as nine semantic features and up to seven syntactic ones per textual attribute.

**Step 5: Matching Scores.** The goal of this step is to estimate the matching likelihood for each pair of candidates based on the feature matrix of the previous step. This is carried out in two steps:

- (i) *Binarizing the feature vectors.* In essence, D-HAT treats each feature as a vote for a “match” (1) or a “non-match” (0) decision. The dimensions of boolean and categorical attributes are already binary. The dimensions of numerical and textual attributes are defined in  $[0, 1]$ , with higher values indicating a higher matching likelihood. To binarize them, D-HAT employs a similarity threshold  $\theta \in [0, 1]$ , common to all dimensions, such that all numeric scores above  $\theta$  are converted into “match” votes (1), while the rest become “non-match” votes (0). All dimensions with a “N/A” value are ignored.
- (ii) *Score estimation.* To calculate the matching score  $m_{i,j}$  for two candidate records,  $r_i$  and  $r_j$ , we aggregate the dimensions of their binary feature vector  $\hat{V}_{i,j}$  into a single value through their mean, i.e.,  $m_{i,j} = \sum_{k=1}^N \hat{V}_{i,j}^k / (N - n)$ , where  $N$  is the total number of features,  $n$  is the number of missing ones and  $\hat{V}_{i,j}^k \in \{0, 1\}$ .

At the end of these two steps, the matching scores of all pairs are calculated and stored in a matrix  $M$ . The records and the matrix define a weighted graph  $G(V, M)$ , where the set of nodes  $V$  represent the input records, and  $M$  is the adjacency matrix of weights.  $G(V, M)$  is referred to as the *similarity graph*.

**Step 6: MutMax Clustering.** The final step receives as input the similarity graph  $G(V, M)$  and partitions it into a set of disjoint clusters, such that every cluster corresponds to a unique entity, containing all duplicate records describing it. The partitioning is performed by **MutMax**, a greedy approach that defines as duplicates the pairs of records with mutually maximum scores. More specifically, MutMax operates as follows: For each record  $r_i$ , all candidates are sorted in decreasing matching scores and the top one  $r_{max}^i = r_j$  is selected as the potential match. If  $r_i$  was set as the potential match for  $r_j$ , the records  $r_i$  and  $r_j$  are designated as matches. The rest of the candidate pairs are ignored.

**Overall approach.** D-HAT algorithm is outlined in Fig. 2. Step 1 (Data cleaning) is applied first (Line 1). Step 2 (Attribute selection) is performed given threshold  $c_{min}$  (Lines 2–7). The overlap blocker is applied to each attribute (Lines 8–10). A performance score is computed per attribute by multiplying the coverage of attribute  $a$  with the reduction ratio [5]:  $getScore(B_a, a) = c(a) \cdot RR(B_a, T)$ , where  $|B_a|$  denotes the total number of candidate pairs in the blocks  $B_a$ . The attribute with the highest score is selected (Lines 11–14), and is applied to retrieve the final set of blocks (Line 16).

The next loop simultaneously applies Steps 4 and 5. It builds a two-dimensional array  $M$  with a score for each pair of compared records. In more detail,  $F \cap a$  is the set of functions applicable for attribute  $a$ . For each feature higher than  $\theta$ , the overall similarity is incremented by one matching vote (Lines 23–25). The average score is finally estimated for the current pair of candidates (Line 29).

Finally, MutMax is applied to  $M$  (Lines 31–36). For each record, the most similar candidate is specified and stored in array  $O$  (Line 31). Using  $O$ , D-HAT identifies the record pairs that are mutually most similar (Lines 32–33), adding them to the output (Line 34). Note that  $D$  is a set and that each output pair is

---

**Algorithm 1** D-HAT

---

**Input:** A data set with duplicates in itself  $T$ , a set of features  $F$ ,  
a threshold on the minimum coverage per attribute  $c_{min}$ , and  
a threshold on the minimum similarity score for binarization  $\theta$

**Output:** The set of duplicate records,  $D$

```

1: Clean  $T$  {Step 1}
2: for all  $a \in T.A$  do {Step 2: attribute selection}
3:   calculate coverage  $c(a)$ 
4:   if  $c(a) < c_{min}$  then
5:      $T.A \leftarrow T.A - \{a\}$ 
6:   end if
7: end for
8:  $best\_attribute = ""$ ,  $best\_score = 0$  {Step 3: blocking}
9: for all  $a \in T.A$  do
10:   $B_a \leftarrow overlap\_blocker(T, a)$ 
11:  if  $best\_score < getScore(B_a, a)$  then
12:     $best\_attribute = a$ 
13:     $best\_score = getScore(B_a, a)$ 
14:  end if
15: end for
16:  $B \leftarrow overlap\_blocker(T, best\_attribute)$ ,  $M \leftarrow \{\}$  {Steps 4-5}
17: for all candidate pair  $(r_i, r_j) \in B$  do
18:    $sim_{i,j} = 0$ ,  $counter = 0$ 
19:   for all  $a \in T.A$  do
20:    if  $r_i.a \neq N/A$  &  $r_j.a \neq N/A$  then
21:      for all  $f_{sim} \in F \cap a$  do
22:         $counter++$ 
23:        if  $\theta \leq f_{sim}(r_i.a, r_j.a)$  then
24:           $sim_{i,j}++$ 
25:        end if
26:      end for
27:    end if
28:  end for
29:   $M[i, j] = M[j, i] = sim_{i,j}/counter$ 
30: end for
31:  $D = \{\}$ ,  $O = \text{argmax}(M)$  per  $r_i \in T$  {Step 6: clustering}
32: for all  $r_i \in T$  do
33:   if  $O[i] = j$  &  $O[j] = i$  then
34:      $D \leftarrow D \cup (i, j)$ 
35:   end if
36: end for
37: Return  $D$ 

```

---

**Fig. 2.** The end-to-end algorithm of D-HAT.

formed with the lowest id in the left part (i.e.,  $i < j$  in  $(i, j)$  in Line 34); as a result, no duplicate pairs are returned as output.

In terms of time complexity, the cost of Steps 1, 2 and 3 is linear with the number of attributes in the given data set  $T$ , i.e.,  $O(|T.A|)$ . For Steps 4 and 5, the cost is  $O(|B|)$ . For Step 6 no sorting is required. Instead, D-HAT merely iterates once over all cells in the two-dimensional array  $M$ . A hash table can be used to store the estimated similarities in practice. As a result, both the time and space complexity of Step 6 (and the entire algorithm) are linear with the number of candidate pairs after blocking, i.e.,  $O(|B|)$ .

## 5 Experimental Evaluation

**Setup.** D-HAT is implemented in Python 3.8.5. All experiments were run on an Ubuntu 18.04.5 server with a 12-core Intel Xeon D-2166NT @2GHz, 64 GB of RAM and 300 GB HDD. A single core was employed in all time measurements.

**Benchmark Data Sets.** We employ five established data sets that come from multiple domains: products, bibliography, restaurants, and healthcare.



**Table 1.** Technical characteristics of the benchmark data sets.  $|S|$ ,  $|T|$  and  $|D|$  stand for the number of source records, target records and duplicate pairs, respectively.

Data set	$ S $	$ T $	$ D $	#Attributes	#Numerical	#Bool. & Cat.	#Textual	#Selected
Amazon-Google	1,363	3,226	1,298	4	1	0	2	3
Abt-Buy	1,081	1,092	1,095	3	1	0	2	3
DBLP-ACM	2,614	2,294	2,223	4	1	4	2	3
Fodors-Zagats	533	331	112	5	0	0	5	5
Immucare	305	310	305	213	32	6	37	75

Immucare is a healthcare dataset matching two hospital visits of the same patient. The technical details of these data sets [13,24] are summarized in Table 1.

**Baseline Systems.** We compare the performance of D-HAT with Magellan [13] and ZeroER [24]. For the former, we use decision tree as the classification algorithm, while for the latter, no configuration is needed.

**Evaluation Measures.** We use the standard measures of recall, precision, and F1-score, which are defined in Sect. 3. We also report the overall run-time, i.e., the time that intervenes between receiving the data set(s) as input and producing the duplicate pairs as output. We repeat every measurement three times and report the average.

### 5.1 Step 3: Blocking

D-HAT applies Magellan’s overlap blocker to all attributes and selects as optimal the one minimizing the number of candidates, while maximizing coverage. The resulting performance appears in Table 2. In all cases, the number of candidate pairs is reduced by whole order of magnitude (i.e.,  $\gg 90\%$ ) in comparison to the brute-force approach (i.e.,  $|S| \times |T|$ ). The only exception is Abt-Buy, where the candidates drop by 86%, which is a dramatic reduction of the search space, too. Nevertheless, the recall in all cases remains rather high, above 90%. This means that the vast majority of duplicate pairs co-occur in at least one block.

Note that precision after blocking remains very low for most data sets. To raise it to acceptable levels, matching is required. Note also that compared to the

**Table 2.** Blocking performance. Time in Seconds.

Data set	Key Attribute	#Candidates	Recall	Prec	Time
Amazon-Google	Name	131,214	0.995	0.010	7.3
Abt-Buy	Name	164,072	0.994	0.007	2.6
DBLP-ACM	Authors	318,404	0.993	0.007	19.4
Fodors-Zagat	Phone	111	0.929	0.936	0.7
Immucare	Date of Birth	311	1.000	0.981	26.5

**Table 3.** Matching effectiveness of D-HAT, Magellan and ZeroER across all data sets. The best F1 per data set is underlined.

Data set	D-HAT									Magellan			ZeroER		
	Syntactic Features			Semantic Features			Hybrid Features			Pr	Re	F1	Pr	Re	F1
	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1						
A-G	0.904	0.479	0.626	0.828	0.349	0.534	0.925	0.532	<u>0.675</u>	0.513	0.573	0.542	0.663	0.385	0.487
A-B	0.818	0.402	<u>0.539</u>	0.635	0.174	0.274	0.824	0.346	0.487	0.440	0.443	0.442	0.220	0.601	0.322
D-A	0.992	0.956	0.974	0.995	0.980	<u>0.987</u>	0.997	0.974	0.985	0.980	0.983	0.981	0.936	0.945	0.940
F-Z	0.981	0.929	<u>0.954</u>	0.971	0.911	0.940	0.981	0.929	<u>0.954</u>	0.939	0.969	<u>0.954</u>	1.000	0.312	0.476
CA	0.993	0.987	<u>0.990</u>	0.990	0.987	0.988	0.993	0.987	<u>0.990</u>	0.968	1.000	0.984	1.000	0.487	0.655

overall run-time of D-HAT and the rest of the methods (in Fig. 3), the overhead of blocking is negligible ( $< 10\%$  in all cases). The only exception is Immucare, where the overhead of blocking is high, due to the very large number of attributes retained after Step 2 (75).

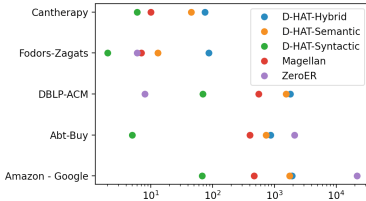
## 5.2 Steps 4–6: Matching

To ensure fairness, we apply the same blocker to the same key attribute for both baseline systems, (e.g., we use the ‘phone’ attribute instead of ‘name’ in Fodors-Zagats). Note that for Amazon-Google, ZeroER could not create its feature matrix within a time limit of 6 h. To complete the assessment, we combined it with the feature vectors created by Magellan instead. As a result, the performance of ZeroER could be slightly different from that reported in [24].

The resulting performance of all algorithms with respect to precision (Pr), recall (Re) and f-measure (F1) appears in Table 3, while the corresponding run-times are reported in Fig. 3. Note that after preliminary experiments, we set  $c_{min} = 0.1$  and  $\theta = 0.7$  for D-HAT in all cases. Note also that D-HAT is combined with three different groups of features: (i) The syntactic ones, which include only the syntactic similarity functions for textual attributes along with the specialized functions of boolean, categorical and numeric attributes. (ii) The semantic features, which differ from the previous group in that they replace the syntactic similarity functions with the semantic ones. (iii) The hybrid features, which employ all similarity functions for all types of attributes defined in Sect. 4. In this way, we are able to examine the contribution of the two types of textual similarity functions, which account for the majority of features used by D-HAT.

Compared to blocking, precision has actually increased by whole orders of magnitude. This emphasis on precision should be attributed to MutMax clustering, which associates every record only with its most similar candidate.

Comparing the various groups of features between them, we observe that the syntactic ones consistently outperform the semantic ones. The reason is that most data sets contain domain-specific terminology. As a result, especially word2vec and GloVe suffer from a large portion of out-of-vocabulary terms. The only exception is DBLP-ACM, which involves long textual attributes like venue



**Fig. 3.** Run-time in seconds.

**Table 4.** The number of features per group.

Data set	Non-textual	Syntactic	Semantic	Hybrid
A-G	4	14	22	32
A-B	4	14	22	32
D-A	4	14	22	32
F-Z	0	35	45	80
CA	78	400	411	733

names and publication titles; in these settings, the evidence provided by semantic similarities outperforms the syntactic ones, albeit by just  $\sim 2\%$ .

In terms of time-efficiency, the advantage of syntactic similarity functions is clear in all cases, as shown in Fig. 3. The run-time of D-HAT increases by a whole order of magnitude in almost all cases, when replacing the syntactic similarity features with the semantic ones. This is caused by the large number of lookups and computations that are required for converting every attribute value into a high-dimensional embedding vector and a similarity score.

It is interesting to examine whether the combination of syntactic and semantic similarities justifies the lower time efficiency by an increase in effectiveness. This is only true in Amazon-Google, where hybrid features’ F1 is higher than the syntactic ones by  $\sim 10\%$ . In all other cases, the hybrid features lie between the two other groups of features, usually closer to the top performing one. Hence, *D-HAT should be exclusively combined with the syntactic group of features.*

Compared to ZeroER, Table 3 shows that D-HAT with syntactic features achieves significantly better effectiveness in most cases. Its f-measure is actually higher by 50%, on average, across the five data sets. At the same time, Fig. 3 demonstrates D-HAT is consistently faster than ZeroER by whole orders of magnitude (e.g., 1 min vs 6 hrs over Amazon-Google) – the sole exception is DBLP-ACM, where D-HAT is slower, due to the computation of 10 syntactic similarity functions over textual values. D-HAT takes into account attributes with high level of noises (missing values, heterogeneity of existing values, errors), which inevitably corrupt some matching signals.

Compared to Magellan, in the first two data sets, D-HAT achieves a higher f-measure than Magellan by more than 13%, while in the next three data sets both methods exhibit practically identical performance (i.e., their f-measures differ by less than 1%). The competitive performance of Magellan stems from its supervised functionality: in each dataset, 70% of the candidate pairs are used for training its classification model, leaving only 30% of the pairs as a testing set. In contrast, D-HAT processes all candidate pairs and its performance is bounded by blocking. In terms of time-efficiency, we observe in Fig. 3 that D-HAT takes a clear lead in all cases, as its run-time is lower than Magellan even by a whole order of magnitude (e.g., 35 vs 400 s over Abt-Buy).

Overall, D-HAT typically outperforms the state-of-the-art unsupervised deduplication method to a significant extent in all respects. Compared to the state-of-the-art supervised approach, it exhibits similar effectiveness, if not higher, at a much lower run-time, despite the lack of labelled instances.

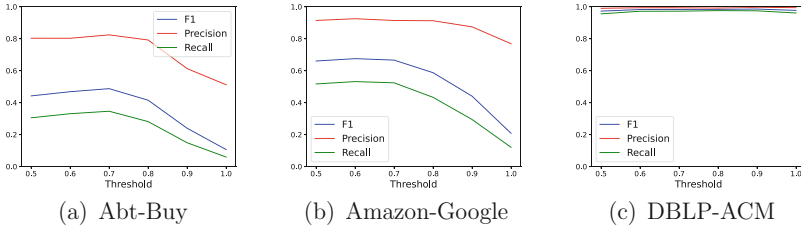


Fig. 4. Performance of D-HAT with syntactic features when varying the threshold  $\theta$ .

### 5.3 Sensitivity Analysis

The only configuration parameter that is crucial for the performance of D-HAT is the similarity threshold  $\theta$ , whose value depends on the level of noise and heterogeneity in the data. To assess its impact on the overall performance of D-HAT, we consider all values in the range  $[0.5, 1]$  with a step of 0.1. The results appear in Fig. 4. Due to lack of space, we report three of the five datasets.

We observe that this parameter has no effect on any evaluation measure over DBLP-ACM. The reason is that the pairs identified as matches in these datasets exhibit very high similarity (practically 1.0) for most of the features employed by D-HAT. As a result, the matching decisions of MutMax clustering are not altered by the value of  $\theta$ . For Abt-Buy and Amazon-Google, we observe that up to 0.7, the performance of D-HAT improves (Abt-Buy) or remains the same (Amazon-Google). For  $\theta > 0.7$ , *a small increase in the similarity threshold yields slightly lower performance with respect to all measures*. The reason is that both data sets are challenging tasks, because they contain many corner cases, i.e., records that are close to the decision boundary.

Overall, we can conclude that D-HAT is robust with respect to its similarity threshold  $\theta$ , with  $\theta = 0.7$  constituting a reliable default value.

## 6 Conclusions

We presented D-HAT, an efficient, fully automated clustering-based end-to-end deduplication system. D-HAT can process high dimensional data sets with heterogeneous attribute types and missing values without requiring user intervention or any labelled data. The thorough experimental study on benchmark data sets demonstrates that our system achieves high accuracy across different benchmark tasks, and outperforms supervised and unsupervised baselines. The main benefit of D-HAT over unsupervised methods is the high accuracy on all standard tasks, whereas compared to supervised methods, D-HAT eliminates the extra time and effort needed from domain experts to annotate a training set. It also saves the time required to find and train an efficient classification model. In the future, we plan to parallelize D-HAT on top of Apache Spark in order to scale it to huge data sets with millions of records.

## References

1. Bojanowski, P., et al.: Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguist.* **5**, 135–146 (2017)
2. Brunner, U., Stockinger, K.: Entity matching with transformer architectures - a step forward in data integration. In: *EDBT*, pp. 463–473 (2020)
3. Chen, M., Mao, S., Liu, Y.: Big data: a survey. *Mob. Netw. Appl.* **19**(2), 171–209 (2014). <https://doi.org/10.1007/s11036-013-0489-0>
4. Christen, P.: The data matching process. In: *Data Matching. Data-Centric Systems and Applications*. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31164-2\\_2](https://doi.org/10.1007/978-3-642-31164-2_2)
5. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.* **24**(9), 1537–1555 (2012)
6. Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., Stefanidis, K.: An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* **53**(6), 1–42 (2021)
7. Devlin, J., et al.: BERT: pre-training of deep bidirectional transformers for language understanding. In: *NAACL-HLT*, pp. 4171–4186 (2019)
8. Dong, X.L., Srivastava, D.: Big data integration. *Synth. Lect. Data Manag.* **7**(1), 1–198 (2015)
9. Fan, W., Ma, S., Tang, N., Yu, W.: Interaction between record matching and data repairing. *J. Data Inf. Qual.* **4**(4), 1–38 (2014)
10. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. *J. Am. Stat. Assoc.* **64**(328), 1183–1210 (1969)
11. Getoor, L., Machanavajjhala, A.: Entity resolution: theory, practice & open challenges. *Proc. VLDB Endow.* **5**(12), 2018–2019 (2012)
12. Hassanzadeh, O., et al.: Framework for evaluating clustering algorithms in duplicate detection. *Proc. VLDB Endow.* **2**(1), 1282–1293 (2009)
13. Konda, P., Das, S., et al.: Magellan: toward building entity matching management systems. *Proc. VLDB Endow.* **9**(12), 1197–1208 (2016)
14. Kusner, M.J., Sun, Y., Kolkin, N.I., Weinberger, K.Q.: From word embeddings to document distances. In: *ICML*, vol. 37, pp. 957–966 (2015)
15. Li, Y., Li, J., Suhara, Y., Wang, J., Hirota, W., Tan, W.: Deep entity matching: challenges and opportunities. *ACM J. Data Inf. Qual.* **13**(1), 1–17 (2021)
16. Liu, Y., et al.: RoBERTa: a robustly optimized BERT pretraining approach. *arXiv preprint [arXiv:1907.11692](https://arxiv.org/abs/1907.11692)* (2019)
17. Mudgal, S., et al.: Deep learning for entity matching: a design space exploration. In: *SIGMOD*, pp. 19–34 (2018)
18. Papadakis, G., et al.: Three-dimensional entity resolution with JedAI. *Inf. Syst.* **93**, 101565 (2020)
19. Papadakis, G., et al.: Blocking and filtering techniques for entity resolution: a survey. *ACM Comput. Surv.* **53**(2), 1–42 (2020)
20. Papadakis, G., et al.: Comparative analysis of approximate blocking techniques for entity resolution. *Proc. VLDB Endow.* **9**(9), 684–695 (2016)
21. Papadakis, G., et al.: The return of JedAI: end-to-end entity resolution for structured and semi-structured data. *Proc. VLDB Endow.* **11**(12), 1950–1953 (2018)
22. Wang, J., Kraska, T., Franklin, M.J., Feng, J.: CrowdER: crowdsourcing entity resolution. *arXiv preprint [arXiv:1208.1927](https://arxiv.org/abs/1208.1927)* (2012)
23. Wang, Z., Sisman, B., Wei, H., Dong, X.L., Ji, S.: CorDEL: a contrastive deep learning approach for entity linkage. In: *ICDM*, pp. 1322–1327 (2020)
24. Wu, R., Chaba, S., Sawlani, S., Chu, X., Thirumuruganathan, S.: ZeroER: entity resolution using zero labeled examples. In: *SIGMOD*, pp. 1149–1164 (2020)