Journal of Big Data

RESEARCH Open Access

Missing values compensation in duplicates detection using hot deck method

Abdulrazzak Ali[1*†], Nurul A. Emran[2†] and Siti A. Asmai[2]

*Correspondence: dowsan1@yahoo.com
†Abdulrazzak Ali and Nurul A. Emran contributed equally to this work
[1] Faculty of Computer and Information Technology, Aden University, Aden, Yemen
Full list of author information is available at the end of the article

Abstract
Duplicate record is a common problem within data sets especially in huge volume databases. The accuracy of duplicate detection determines the efficiency of duplicate removal process. However, duplicate detection has become more challenging due to the presence of missing values within the records where during the clustering and matching process, missing values can cause records deemed similar to be inserted into the wrong group, hence, leading to undetected duplicates. In this paper, duplicate detection improvement was proposed despite the presence of missing values within a data set through Duplicate Detection within the Incomplete Data set (DDID) method. The missing values were hypothetically added to the key attributes of three data sets under study, using an arbitrary pattern to simulate both complete and incomplete data sets. The results were analyzed, then, the performance of duplicate detection was evaluated by using the Hot Deck method to compensate for the missing values in the key attributes. It was hypothesized that by using Hot Deck, duplicate detection performance would be improved. Furthermore, the DDID performance was compared to an early duplicate detection method namely DuDe, in terms of its accuracy and speed. The findings yielded that even though the data sets were incomplete, DDID was able to offer a better accuracy and faster duplicate detection as compared to DuDe. The results of this study offer insights into constraints of duplicate detection within incomplete data sets.

Keywords: Duplicates detection, Incomplete data set, Clustering, Sorting key, Compensation method

The need for duplicate detection

Duplicate record is one of the most common data quality problems especially within huge volume databases. Thus, the process of removing duplicates or known as 'deduplication' becomes essential in many applications. Data deduplication is significant in avoiding substantial biases during data analysis [1], reducing the storage space (and its costs), and minimizing the pressure of input/output on the storage system [2, 3]. Duplicates are also removed to reduce network traffic at the expense of elevated computational overhead. From the energy-saving perspective, deduplication creates a balance between the energy consumed for the additional computation and the energy provided by low storage and network load [4, 5]. Duplicate detection process should be conducted prior to deduplication process.

Duplicates are detected by searching for all objects (or records) that represent the same real-world entity to ensure database consistency [6]. A duplicate detection mechanism tends to search similarities between two or more entities by using similarity measures. The similarity measure is based on equivalence relation where, suppose the real-world entities are *er*, *es*, and *et*, if $er \equiv es$ and $er \equiv et$ then $es \equiv et$. Because of this transitive relationship, the duplicate detection will be partitioning all database entities into non-empty and disjointed partition classes where each system refers to another real-world entity. Besides, an entity is represented as a relation in a relational database.

Although clustering is the result, decisions upon the presence of duplication are often made in a pairwise fashion before they are combined with a globally consistent result, namely duplicate clustering. The pairwise comparison approach relies on the use of similarity in attribute values to indicate real-world equivalence between the corresponding database entities. According to the score of similarity between the attribute values during the candidate pair comparison, each entity pair is assigned to the MATCHES class (assumed duplicates) or the UNMATCHES class (no duplicates are assumed). The similarity measure can calculate the similarity for the attributes individually and aggregate their similarity to an overall record similarity. If necessary, a domain expert might be called to verify the similarity results.

In general, duplicate detection methods are prone to two types of error: (a) false-positives: detection of the actual non-duplicates as duplicates, and (b) false negatives: no detection of the actual duplicates [7]. Relaxing the matching criteria increases the number of false positives while tightening them may increase false negatives. There is usually a trade-off between both types of errors in duplicate detection. Some applications consider that false positives are worse than false-negative which results in their effect on the accuracy of the detected duplicates, whereas the opposite is true with other applications [8].

The challenge of detecting duplicates

Detecting duplicates is challenging especially within incomplete data sets. In this case, pairwise matching tends to be unsuccessful without a pre-processing step. Such pre-processing step is removing attributes with high rates of missing values or replacing missing values with estimated values based on the data set existing values (imputation) [9, 10]. Besides, before matching can be performed, attribute selection must be conducted. The desired high matching ratio (that relies on the results of attribute selection) is crucial in determining valid string comparison at a later stage.

Detecting duplicates within incomplete data sets pose a unique challenge [11–15]. This is because missing values that are present within records will make these records look unique although they refer to the same real-world entity. As a result, several records (the duplicates) of the same real-world entity are kept and treated as if they are different. Therefore, the challenges facing the duplicate detection process within incomplete data sets can be summarized as follows:

- Data clustering: In duplicate detection, the data sets are partitioned into small groups or clusters to reduce the time for pairwise comparisons. Clustering incomplete data sets is difficult especially if the attributes used to create a sorting key contain missing values. As a consequence, the probability of the records (that are likely to be similar) being grouped will be reduced. Thus, candidate attributes that can be used to create the sorting key based on property uniqueness and completeness need to be identified accurately. Candidate attributes refer to the attributes with low repetition value rate and low missing value rate. The selection of these attributes determines the success of duplicate grouping. However, the success ratio depends on the number of missing values of the selected attributes. Therefore, the process of compensating the missing values in the selected attributes is necessary to increase the probability of having records that refer to the same entity of the real world in the same group [16].

- Data matching: This process is important in finding duplicate records in a data set. The matching process is performed by comparing the values of the attributes in several ways such as by character to character, token to token, or sound to sound. The standard matching methods usually fail when the data sets under consideration are incomplete, especially in the case involving nominal values [17].
  Comparing all attribute values is expensive. Therefore, to find a suitable string (called a token) from the attributes to increase the probability of matching records is deemed crucial. The creation of the comparison strings with the same mechanism of the selected attributes to form sorting keys may seem logical. Although a large number of presented methods for duplicate detection exist, there is still a need for substantial improvement to deal with a huge number of errors daily [18]. Therefore, this calls for a new method to address the problem of duplicate record detection within incomplete data sets.

Full duplicate detection is often unattainable, and whether the false-positive results are preferred more than the false negatives depends on the requirement of database applications. An independent implementation process for duplicate detection algorithm needs to adjust the values of the input parameters, such as values of similarity measures or threshold [19–21]. These values have to be adapted according to the given detection scenario. Thus, the configuration of duplicate detection depends heavily on the domain under study and some quality characteristics (such as completeness or accuracy) for the attributes within the data sets.

The rest of this paper is structured as follows. In the Related Work section, the problem of missing values and the work related to duplicate detection techniques are presented. The details on the mechanism of the proposed method for detecting duplicates within incomplete data sets are provided in the Duplicate Detection within Incomplete Data Sets (DDID) section, followed by a section on Experimental Setup, Results, Discussion and Conclusion.

Related work

The problem of duplicate detection has been addressed under different names such as record linkage, entity matching, record reconciliation, or merge/purge. In general, the proposed algorithms for duplicate detection are commonly aimed to achieve:

1. The efficiency of duplicate detection process by reducing the number of comparisons for candidate pairs. In this case, the speed of the algorithm is of concern and influenced by the number and cost of comparisons; and

2. The effectiveness of duplicate detection to classify candidate pairs of records as duplicates or non-duplicates accurately. In this case, the accuracy of the algorithm is of concern [22, 23].

Elmagarmid et al. presented a survey of duplication detection techniques to detect duplicate entries that are non-identical in the data set [18]. The survey indicated that there is no clear vision in determining which measures and methods are the current state-of-the-art. The survey attributed this problem to the lack of standardized data and large scale of benchmarking data sets to compare new methods with the existing ones. Duplicate detection methods have been categorized into two types: (1) methods that rely on the training data to match the data records by using probabilistic approaches and supervised machine learning techniques to develop matching techniques, and (2) methods that rely on domain knowledge or generic distance metrics to match data records. Köpcke and Rahm conducted a comprehensive overview of the current detection of duplicates [24]. They compared eleven frameworks and distinguished three types of frameworks namely training-based frameworks, frameworks without training, and hybrid frameworks. Several criteria were used for comparison such as the types of supported entities and methods of data partitioning to reduce the search area. "Matcher” is the term used for the algorithm that is used to determine if two entities are similar enough to represent the same real-world entity. It also examines the possibility to combine multiple matches and use training data if necessary (see [25, 26]). In general, researchers are aware of the difficulty of detecting duplicates within incomplete data sets [12, 18].

Incomplete data sets due to missing values have been reported as one of the challenges in data duplication [10, 27], along with other issues. These issues are typographical errors, abbreviations, and different representations of the same logical values [12, 23, 28–31]. Records with missing entries are of two forms: fully missing (all attributes’ values except the primary key are missing) or partially missing (some of the attributes’ values are missing). In the case of fully missing, as information about the record is missing entirely, the probability to label it as a duplicate is lower as compared to the partially missing case. In reality, fully missing is a rare case. Detecting duplicate records with partially missing values is difficult especially in the case where a real-world entity is accidentally represented by multiple key attribute values in a data set (i.e., as a result of data integration) or in the case where the key attribute is missing (or unknown) [32].

Missing value problem is usually handled either by predicting the value or by disregarding the missing value completely [33]. The most common methods to deal with the strings that contain missing values can be categorized as follow:

1. Deletion: A completely missing point is skipped when utilizing a data set with the missing value [34].
2. Interpolation: In a geometric sense, a line is drawn between the ends of sequences on either side of the missing value [16].
3. Imputation: This operation involves fixing of value at the point where it is missing in the string using a specific algorithm like K-means, where the missing value will be replaced with a substituted value.

4. Smoothing: Smoothing is applied in estimating missing values in time series data for univariate data. Traditional time series analysis is commonly directed toward scalar-valued data and represented by moving average, or autoregressive-moving average models to smooth them into one continuous curve.

Incomplete data sets have been reported as an obstacle in the data clustering phase [10], as missing values can affect the creation of the sorting keys. Besides, the presence of missing values leads to failure in the matching process due to two factors. Firstly, the rate of missing values in the records to match. If the selected attributes contain a high rate of missing values, a high rate of false-negative and false-positive will be produced. Secondly, the type of attributes that form comparative strings "Tokens". For example, if the "Token" is created from the attributes whose value repetitiveness is high (such as "city" or "district"), the record pair will be identified as duplicates even though they are not.

Several duplicate detection methods highlight the issue of compiling and partially comparing complete records. The problem of incomplete data that can be found in data sets takes several patterns (see [35, 36]).Missing value is a type of data completeness issue that affects duplicate detection. Several methods as mentioned earlier are proposed to deal with the problem of missing values in data sets, such as through ignoring records with missing entries, manually imputing missing data, and using the expectation-maximization (EM) algorithm. In cases where there are a large number of missing entries, or the data sets contain a large number of attributes where the probability of missing at least one entry is high, the first two methods are not practical. Moreover, it has been reported that a method such as EM is impractical to estimate missing data within a large number of attributes (i.e., the feature set is large), being computationally difficult as well as the algorithm convergence rate (EM) will be very slow if the missing rate is high (a large portion of the data is missing) [32, 37–39].

Tamilselvi and Saravanan presented a model for duplicate detection (and elimination) caused by errors and missing values [23]. The proposed model is based on several steps:

1. Use an attribute selection algorithm for the best attribute for duplicate identification;
2. Convert the selected attribute value to a token;
3. Partition data by a clustering algorithm;
4. Compute match rate by using similarity measures; and
5. Remove duplicates and merge.

In this model, the completeness criterion for the selected attributes relies on token matching.

Van Gennip et al. proposed a method for extending the TF-IDF soft method to address the two common issues in the duplicate detection method namely, the sparsity (that occurs due to missing entries) and having more than one instances for the same object [32]. The proposed method is based on: (1) calculating similarity scores between records, (2) grouping records together into independent groups, and (3) comparing different methods to generate similarity scores between records (plus a different set of string matching, frequency-inverse methods, and ngram techniques). They also suggested the possibility of replacing the method of dealing with missing values in both TF-IDF and Jaro-Winkler by performing a data imputation with a potential candidate.

Nevertheless, duplicate detection methods still need further refinement to overcome obstacles that prevent the detection of records that are likely to be similar. Duplicate detection methods are found to be limited in handling incomplete data sets in the following aspects:

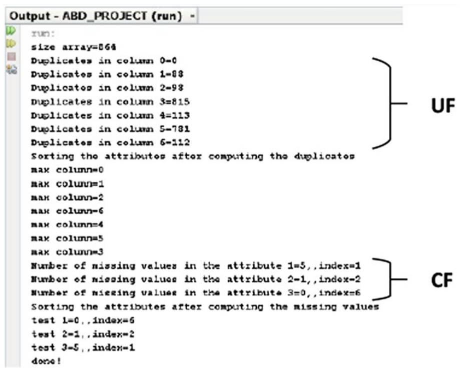
1. Lack of an optimal mechanism to avoid the problem of missing values during the attribute selection using appropriate criteria for creating the sorting keys.
2. No appropriate compensation mechanism for selected attribute missing values.
3. Improvement of duplicate detection time and precision matching.
4. Lack of intuitive method for using duplicate detecting tools that require less user's experience and expertise.

Duplicates detection within the incomplete data sets (DDID)

In this paper, Duplicate Detection within the Incomplete Data sets (DDID) was proposed to address the limitation of existing works. DDID was inspired by a framework of "DuDe" for duplicate detection toolkit (See [40]). Several stages of duplicate detection were extended in DDID to address the problem of missing values that affected the duplicate detection processes. These stages involved automatic attribute selection by using criteria (uniqueness and completeness) that would ensure the quality of the selected attributes. A dynamic mechanism was adopted to create the sorting key from the selected attributes for each record in the data set for the clustering stage. Finally, comparative strings were created for the matching stage.

The attribute selection phase was the cornerstone for detecting duplicates within incomplete data sets of the DDID method. In this stage, the "appropriate" attributes were defined for generating the sort keys for the clustering stage and the matching strings to calculate the similarity. The attribute selection process depended on the rate of duplicate values and missing values in each attribute as well as the threshold value of the acceptance rate. The attribute selection process underwent three main sequential stages as shown in Algorithm 1. The goal of the attribute selection algorithm was to reduce the time spent by data specialists in identifying appropriate attributes for detecting duplicates and increasing the speed in the later stages. The goals were achieved by avoiding selecting attributes that increased the number of unnecessary comparisons (because of the missing values or the high rate of duplicates values in it). The algorithm began by defining the value of the threshold T, where $0 > T > 1$. The higher the value of T the greater the acceptability of the amount of repetition and the missing values in the attribute. Good selection of the threshold value is essential because choosing a very low threshold value affects the creation of the sort key, which, in turn, affects the detection of duplicates in the data set.

```
Algorithm 1: Attributes selection algorithm
Input: $N$ Attributes, $n$ Data set size, $T$
Output: $S$ Subset of attributes
Var: $A$ Attribute set, $i, j$
begin
  1  $threshold \leftarrow T$
  2  Unique $(U)$ value of the attribute $Ai$ if row $R_{i_1^n}$ $R = Ri$
  3  Missing $(C)$ value of the attribute $Ai$ if row $R_{i_1^n}$ $Ri = Null$
  4  Calculate the average of $U$ and $C$ for $Ai$ to $n$.
  5  Compare $(T, AVG)$
  6  Rank $(N, AVG)$
end
Compute:
  1  $U_j = n-$ Count(Distinct $A_j$)
  2  $C_j = \frac{\sum_{i=0}^{N-1} completeness_{i,j}}{n}$
  3  $AVG = \frac{n-(U_j+C_j)}{n}$
```



```
run:
size array=864
Duplicates in column 0=0
Duplicates in column 1=88
Duplicates in column 2=98
Duplicates in column 3=815
Duplicates in column 4=113
Duplicates in column 5=781
Duplicates in column 6=112
Sorting the attributes after computing the duplicates
max column=0
max column=1
max column=2
max column=6
max column=4
max column=5
max column=3
Number of missing values in the attribute 1=5,,index=1
Number of missing values in the attribute 2=1,,index=2
Number of missing values in the attribute 3=0,,index=6
Sorting the attributes after computing the missing values
test 1=0,,index=6
test 2=1,,index=2
test 3=5,,index=1
done!
```

Fig. 1 Implementing the uniqueness and completeness functions on a restaurant dataset

The uniqueness factor in the attribute selection is a decisive factor in limiting unnecessary comparisons due to the high frequency of some attribute values such as (gender) whose values are within a specified range, which may lead to a high rate of false-positive. The uniqueness function (UF) counts the number of duplicates in each attribute by using a HashSet class. Attributes are arranged according to the number of duplicates in ascending order. In DDID, the sort keys and matching strings were created from three attributes, hence, the completeness function (CF) calculated the numbers of missing values in the first three (low repeated) attributes after excluding the primary key attributes. Figure 1 shows the result of applying the uniqueness and completeness criteria to the restaurant data set.

Nevertheless, as there is a possibility for the presence of missing values in the selected attributes that eventually will cause a clustering problem, the missing value problem needs to be resolved first. Therefore, the use of an appropriate compensation method for missing values is necessary to improve the duplicate detection.

Missing values compensation

To a large extent, the compensation method such as Hot Deck is generally acceptable to deal with missing values, especially when the record has more than one occurrence in the data set. In the Hot Deck imputation, missing values are imputed from other records in the database that share attributes related to the incomplete attribute [41].

As indicated by the Hot Deck strategy, the missing value for the record is compensated by searching for the missing value in the complete records that are most similar to the record that consists of missing values [42, 43]. This technique depends on the correlation of influencing values where matching of more specific attributes is used. Hence, missing values are imputed from other records in the database that share attributes related to the incomplete records.

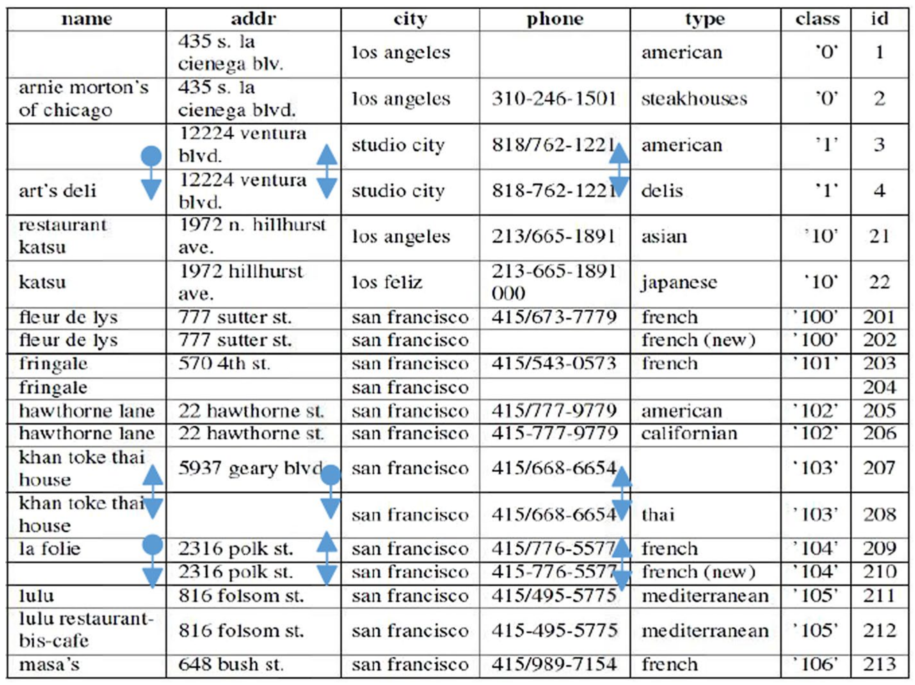
In DDID, the Hot Deck imputation method was used to compensate the missing values in the high-rank attributes that were specified in the attribute selection stage to avoid any defect in the creation phase of the dynamic sorting key and comparative strings. Listing 1 shows the method used to compensate for the missing values in the high-rank attributes.

```
void compen(){
    String first="";
    String second="";
    int index1, index2, i ,j ,m;    int l;  int nex; int ep;
    \\selected attributes sorted ascending
    \\ searching for missing values
    for each selected attribute of [i]{
        for each row of [j]{
            if(R[j][i] contains "" or "null")
            {
                add the index of the other selected attributes
                     to (index1 and index2)
                add the values of the other selected
                    attributes to (A and B)
                if(A and B contains "" or "null")
                Mark record for next process("Avoid
                    comparisons" and "Remove candidate")
                else{
                    for each selected attribute of [i]{
                     for each row of [m]{

                        if(R[m][index1] contains(A)){
                            if(R[m][index2] contains(B)){

                                R[j][i]=[m][i];

                                Break;
                }}}}}}}}
```



name	addr	city	phone	type	class	id
	435 s. la cienega blv.	los angeles		american	'0'	1
arnie morton's of chicago	435 s. la cienega blvd.	los angeles	310-246-1501	steakhouses	'0'	2
	12224 ventura blvd.	studio city	818/762-1221	american	'1'	3
art's deli	12224 ventura blvd.	studio city	818-762-1221	delis	'1'	4
restaurant katsu	1972 n. hillhurst ave.	los angeles	213/665-1891	asian	'10'	21
katsu	1972 hillhurst ave.	los feliz	213-665-1891 000	japanese	'10'	22
fleur de lys	777 sutter st.	san francisco	415/673-7779	french	'100'	201
fleur de lys	777 sutter st.	san francisco		french (new)	'100'	202
fringale	570 4th st.	san francisco	415/543-0573	french	'101'	203
fringale		san francisco				204
hawthorne lane	22 hawthorne st.	san francisco	415/777-9779	american	'102'	205
hawthorne lane	22 hawthorne st.	san francisco	415-777-9779	californian	'102'	206
khan toke thai house	5937 geary blvd	san francisco	415/668-6654		'103'	207
khan toke thai house		san francisco	415/668-6654	thai	'103'	208
la folie	2316 polk st.	san francisco	415/776-5577	french	'104'	209
	2316 polk st.	san francisco	415-776-5577	french (new)	'104'	210
lulu	816 folsom st.	san francisco	415/495-5775	mediterranean	'105'	211
lulu restaurant-bis-cafe	816 folsom st.	san francisco	415-495-5775	mediterranean	'105'	212
masa's	648 bush st.	san francisco	415/989-7154	french	'106'	213

Fig. 2 An example of missing values compensation

The similarity of the selected attributes for the two records during the compensation process was a condition for the compensation procedure. To illustrate this, a simplified case for the data set is as shown in Fig. 2. The selected attributes were *name*, *addr* and *class*, where record 3 and 4 were regarded as similar. Therefore, the missing *name* value in record 3, was compensated by the value 'art's deli' taken from the *name* value in record 4; as records 207 and 208 were similar, the *addr* value which was missing from record 208 was compensated by *addr* value from record 207. Details on the similarity algorithm used are presented in the Experimental Setup section.

Data sets

The data sets used in this study were gold standard data sets, available on the Hasso Plattner Institute (HP) website under the Duplicate Detection Project (DuDe). These data sets have been frequently used in duplicate detection research works [40, 44–53]. In addition, the MusicBrainz data sets (as reported in [54]) were also used in this study.

Restaurant data set

This data set was originally taken from Fodor's and Zagat's restaurant guides. This data set consisted of attributes namely name, address, city, phone, type, and class. The data set comprised 866 records of which 13% of them were duplicates.

CD data set

This data set was bigger than the Restaurant data set as it contained 9763 records and 107 attributes. CD-related attributes such as artist, title, category, genre, cdextra, year, and track were kept in this data set. Only 3% of the records were duplicates.

These two data sets underwent a series of changes as reported in Draisbach and Naumann (see [40]) in addition to the changes mentioned in [15]. The unique identifiers that were added by the authors had been removed so that the creation of the dynamic sorting keys was not affected by them, as these identifiers did not belong to the actual data sets.

MusicBrainz

These data sets were generated using the data pollution tool called as DaPo [54]. The first data set of MusicBrainz contains 193,750 tuples while the second data set contains 800,000 tuples extracted from 1,937,500 tuples, where each tuple described a certain audio recording. These data sets consisted of 12 attributes such as TID, CID, CTID, SourceID, id, number, title, length, artist, album, year, and language.

Evaluation of DDID

To evaluate DDID, an experimental approach was adopted. We hypothesized that even though under the constraint of incomplete data set, DDID was capable to accurately detect duplicate records. DuDe, an earlier duplicate detection method, was used as a benchmark to evaluate DDID's performance. In addition to accuracy, the speed of detection was also observed. The following measures were used in the experiment:

1. Duplicates detection accuracy: To evaluate the accuracy of DDID in detecting duplicates within incomplete data sets, the measure of Draisbach and Naumann's (2010) study (refer [40]) was used. The creation of a statistical component was used to read the file containing the gold standard. Duplicate pairs from the transitive closure and non-duplicate pairs from the comparison stage were added to the statistical component and compared with the gold standard. The statistical component calculated the number of true-negatives and false-negatives of the actual classified pairs. The statistical component needed non-duplicate pairs to obtain an additional measure for the quality of the comparison. The accuracy of DuDe and DDID performance was determined by calculating the F-Score that combined recall and precision through the harmonic mean. Thus, it is a measure of the accuracy of the experiment [22, 55]. The Recall, Precision, and F-Score were computed by using certain formulas:

$$Recall = \frac{|true\,positive|}{|true\,positive| + |false\,negative|} = \frac{|true\,positive|}{|true\,duplicates|} \tag{1}$$

$$Precision = \frac{|true\,positive|}{|true\,positive| + |false\,positive|} = \frac{|true\,positive|}{|declared\,duplicates|} \tag{2}$$

$$\text{F-Score} = \frac{2 \times recall \times precision}{recall + precision} \tag{3}$$

The formula construct descriptions are as follows:

- False positive: Represents the number of candidate pairs, wrongly declared as duplicate records.
- False negative: Represents the number of candidate pairs, wrongly declared as non-duplicate records.
- True positive: Represents the number of candidate pairs, correctly declared as duplicate records.
- True negative: Represents the number of candidate pairs, correctly declared as non-duplicate records.

2. Statistical analysis: was used to determine whether the performance of DDID was statistically significant from DuDe.

Experimental setup

To experiment with duplicate detection within an incomplete data set, an arbitrary pattern had been used to hypothetically add missing values randomly within the data sets under study. 4% missing values were added in the key attributes of the Restaurant data set whereas 1.5% missing values were added in the CD data set. For the MusicBrainz data sets, the missing values were not injected due to the high percentage of missing values (blank, null, and unknown) from the source. Table 1 shows the rate of missing values (in percentage) in the key attributes for two MusicBrainz data sets. The data sets were stored in CSV format. The experiment was conducted with Java 8.2 installed on Windows 10 platform using a Dell laptop with 4GB RAM, Core i3 processor, 500GB HDD.

Figure 3 illustrates the steps adopted in the experiment. Once the data sets were ready, a Java class that had been created to read incomplete data sets was run. For DDID, attribute selection routines, known as uniqueness and completeness functions, were used to define high-ranked attributes before dynamic sort keys could be generated. Next, the missing values were compensated by the compensation algorithm (using the Hot Deck imputation method). In addition, the missing values compensation was only performed by DDID, instead of DuDe. However, the steps afterward were the same for both DDID and DuDe.

The sorted neighborhood algorithm (see [56]) with a window size of 20 records was used to sort the keys for Restaurant data set with the size of 50 records for CD data set and 150 records for MusicBrainz data sets. Levenshtein Distance similarity algorithm (see [57]) with a similarity threshold set to 0.9 was used to measure the similarity between the comparative strings in DDID and the chosen attributes in DuDe. The major difference between DuDe and DDID was in the sort keys generation. DuDe depended on

Table 1 Missing values rate in MusicBrainz data sets

Data Set	Title (%)	Artist (%)	Album (%)
MusicBrainz(A)	0.67	21	24
MusicBrainz(B)	0.68	21	29

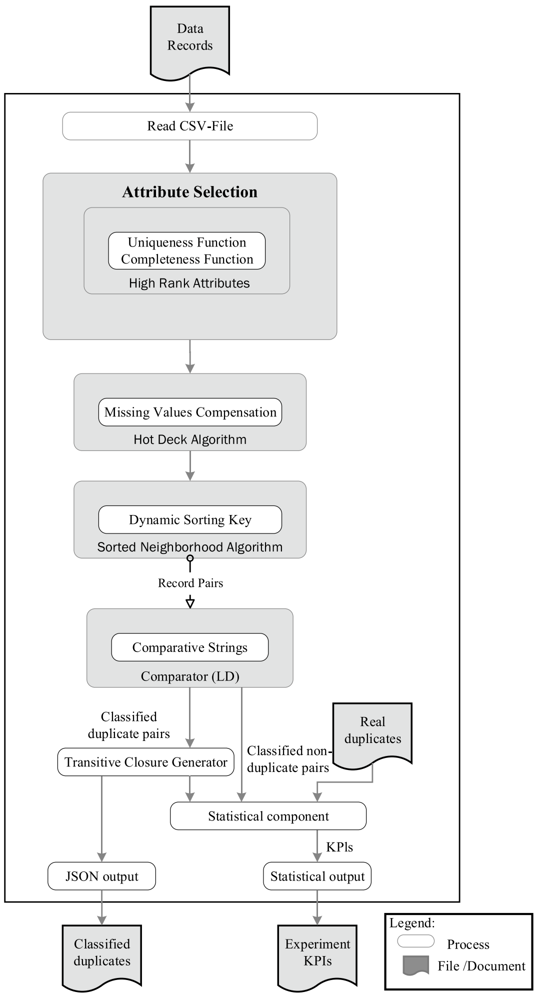


Fig. 3 Experiment work-flow

the attribute chosen by the users as the value for the sort keys, such as the name attribute in the Restaurant data set and the artist attribute for both CD and MusicBrainz data sets. In contrast, DDID was based on the fulfillment of criteria for the uniqueness and completeness functions in the selected attributes to dynamically generate the sort keys and also the comparative/matching strings. The sort keys in DDID combined a portion of the selected attribute values (e.g., the first three elements of the value of each selected attribute) together into a single string, where the same applied to comparative/matching strings with a certain length (e.g., the first eight elements of the value of each selected attribute as used with CD data set).

The results of duplicate pairs from the transitive closure and non-duplicate pairs from the comparison stage became the input for the statistical component generation. Key outputs such as the detection runtime, number of generated record pairs, and number of classified duplicates were calculated. Error types in duplicate detection (false-positives, false-negatives, true-positives, and true-negatives) were also measured where the gold data was used as reference. The final key performance indicators (KPIs), namely Precision, Recall, F-measure, and Accuracy were calculated by using the statistical outputs.

Table 2 Experiment results

Data set	Method	Declared duplicates	True positives	False positives	False negatives	True negatives	Precision	Recall	F-measure
Restaurant	DuDe	97	78	19	34	16114	0.80	0.70	0.75
	DDID	97	97	0	15	16133	1.00	0.87	0.93
CD	DuDe	306	205	101	94	476762	0.67	0.69	0.68
	DDID	262	236	26	63	476837	0.90	0.79	0.84
MusicBrainz (A)	DuDe	97090	33103	63987	110647	28649838	0.34	0.23	0.27
	DDID	79818	52185	27633	91565	28686192	0.65	0.36	0.46
MusicBrainz (B)	DuDe	224125	46414	177711	216280	118748569	0.20	0.17	0.18
	DDID	216722	53757	162965	208937	118763315	0.25	0.20	0.22

The statistical outputs were stored in CSV, whereas the classified duplicates were kept in a JSON file.

Results and discussion

Key performance results

Table 2 shows the results of the key outputs yielded in the experiment from the implementation of DuDe and DDID. Duplicate detection by both methods involved 16,245 candidate pairs for the Restaurant data set and 477,162 candidate pairs for the CD data set and 28857575 candidate pairs for MusicBrainz (A) data set and 119188974 for MusicBrainz (B) data set.

Both methods showed the same number of declared duplicates for the Restaurant data set, however, for the bigger data sets which are CD, MusicBrainz (A), and MusicBrainz(B), DuDe declared more duplicates than DDID. For all data sets, DDID showed a better performance than DuDe by producing fewer errors.

Figure 4 shows the results for the key performance indicators namely the Recall, F-score, and Precision by both methods. Overall, the scores of DDID were higher than DuDe in all categories.

It is important to point out that the procedures used in DDID did not only increase the efficiency of the duplicate detection within incomplete data sets. Therefore, to clarify the behavior of DDID in dealing with duplicates within the complete data sets, both methods were tested on the completed restaurant data set before adding the missing values. Table 3 showed that DDID performed well in detecting duplicates within the complete data set as compared with DuDe.

In the experiment, the value of the elapsed time for duplicate detection was represented by milliseconds for both DuDe and DDID. A statistical analysis was conducted using the SPSS to determine the significance of performance demonstrated by DDID against DuDe. As shown in Fig. 5, since the result of the data distribution for the elapsed time for both DDID and DuDe was normally distributed, a parametric test, the *t*-test was used.

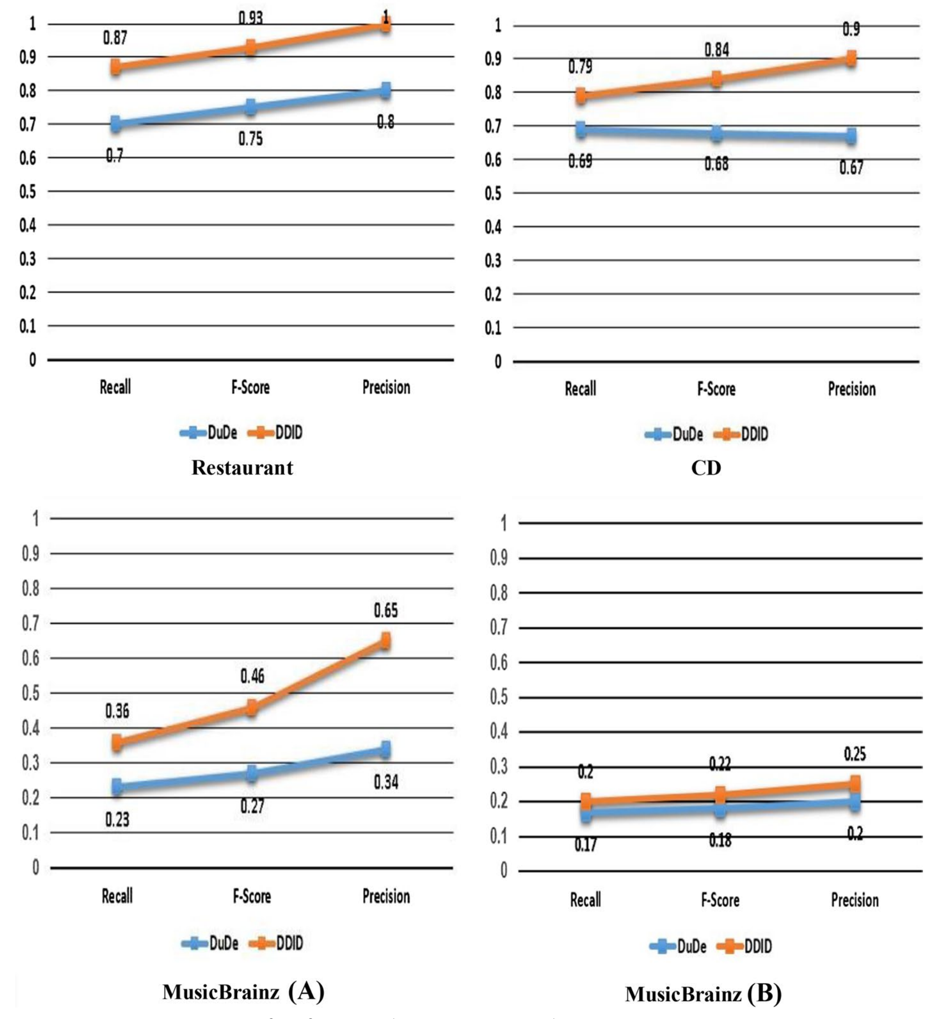


Fig. 4 Accuracy comparison of performance between DDID and DuDe

Table 3 Implementation result on the complete restaurant data set

Data set	Method	Declared duplicates	True positives	False positives	False negatives	True negatives	Precision	Recall	F-measure
Restaurant	DuDe	92	84	8	28	16106	0.91	0.75	0.82
	DDID	98	98	0	14	16114	1.00	0.88	0.93

Hypotheses used in this test were as follows:

- Null hypothesis, H_0: The mean elapsed time for both DuDe and DDID is the same.
- Alternative hypothesis, H_a: The mean elapsed time for both DuDe and DDID is not the same.

Table 4 shows the results of the t-test on the elapsed time of DuDe and DDID. In the t-test results, DuDe had a larger mean than DDID with $M = 2317.90$ ms ($SD = 184.628$), $M = 11945.07$ ms ($SD = 574.791$), $M = 1083948.73$ ms ($SD =$

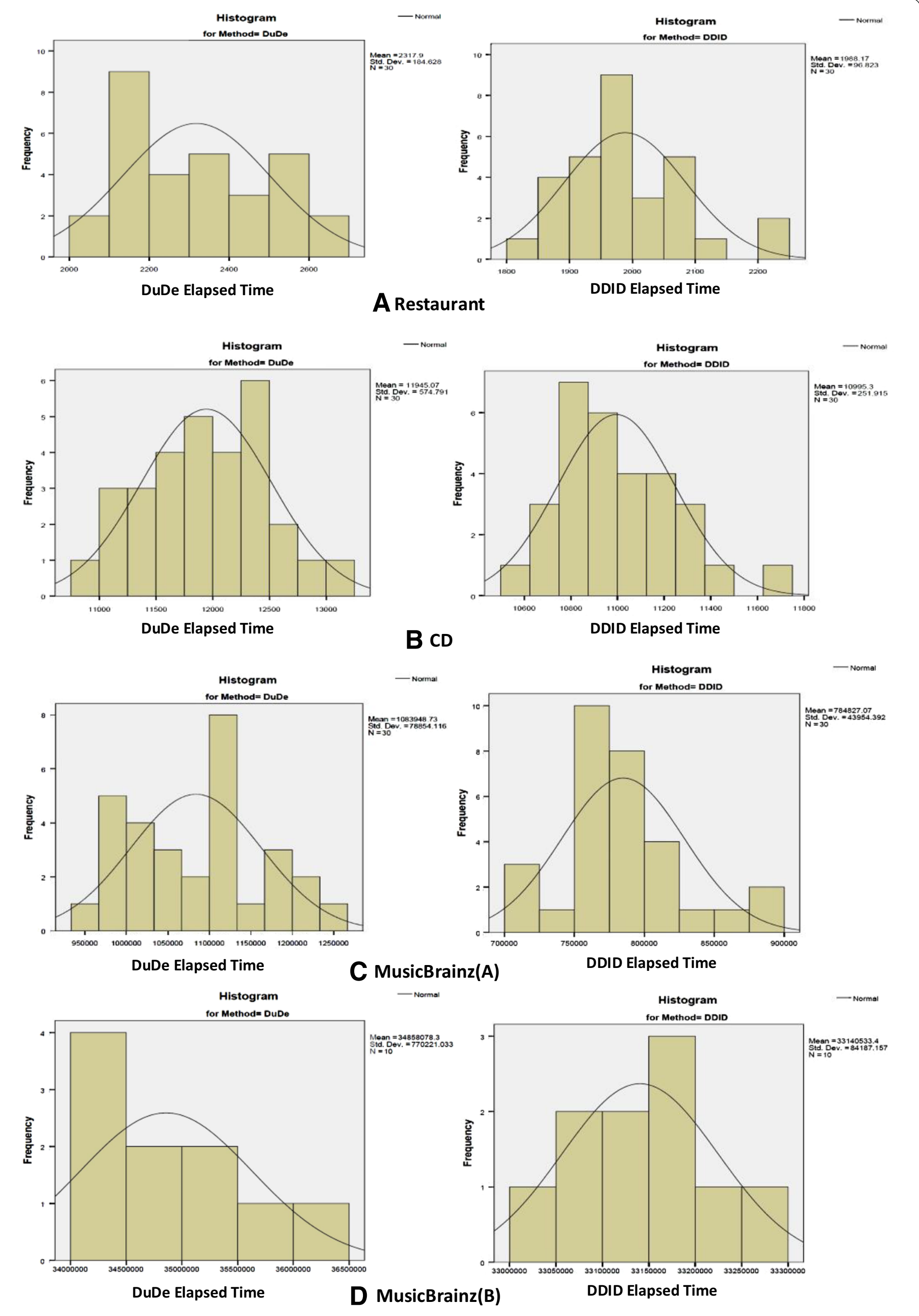


Fig. 5 Data distribution

78854.116), and $M = 34858078.30$ ms ($SD = 770221.033$) for the Restaurant, CD, MusicBrainz(A) and, MusicBrainz(B) data sets, respectively.

As $P < 0.05$, the null hypothesis was rejected and the alternative hypothesis was accepted. Thus, there was a significant difference in terms of duplicate detection speed between DuDe and DDID for the data sets.

Moreover, the elapsed time factor was directly related to the following:

Table 4 Independent samples *T*-test for elapsed time

Data set	Method	P-value	Mean	Std. deviation	t	95% confidence interval of the difference	
						Lower	Upper
Restaurant	DuDe	0.000	2317.90	184.628	8.663	253.543	405.923
	DDID	0.000	1988.17	96.823		253.015	406.451
CD	DuDe	0.000	11945.07	574.791	8.289	720.413	1179.120
	DDID	0.000	10995.30	251.915		718.149	1181.385
MusicBrainz(A)	DuDe	0.000	1083948.73	78854.116	18.148	266128.812	332114.522
	DDID	0.000	784827.07	43954.392		265933.426	332309.908
MusicBrainz(B)	DuDe	0.000	34858078.30	770221.033	7.010	1202785.593	2232304.207
	DDID	0.000	33140533.40	84187.157		1165245.592	2269844.208

1. Both methods used sorted neighborhood algorithm, where the bigger size of window, the larger number of comparisons, hence, increasing duplication detection elapsed time.
2. The size of the data within the attributes of the data set: the larger the data size, the greater the number of comparisons, and since the duplicate detection model or DDID depended on specific parts of the values of the selected attributes to generate matching strings, the elapsed time would be lower.
3. The presence of a high rate of missing data, as in the case of the MusicBrainz data sets, led to an increase in the false-positive, thus, increasing the elapsed time. In DDID, this problem was reduced by using the compensation algorithm (Hot Deck). In addition, the sort keys and matching strings were created from multiple attributes, hence, reducing the possibility of missing value presence.

From the results, the Hot Deck method used in the DDID method can improve the performance of elapsed time and accuracy in duplicate detection. More importantly, the improvements made by DDID are statistically significant. At the same time, the percentage of missing values negatively affects the accuracy of duplicate detection if it forms a large fraction of the key attribute values.

Conclusion

In conclusion, Duplicate Detection within the Incomplete Data sets (DDID) method has been proposed to encounter missing value problem in duplicate detection. Attribute selection mechanism is introduced in DDID where the uniqueness and completeness criteria are fundamental elements for sorting key creation and string comparison. To reduce the impact of missing values on the selected attributes of duplicate records, a compensation method has been included using the Hot Deck algorithm to compensate for missing values in the selected attributes. Then, the accuracy and speed of duplicate detection of DDID have been evaluated against DuDe which is used only with complete data sets, instead of data sets with missing values.

The results show that DDID behaves better than DuDe in all aspects under study. In addition, the Hot Deck compensation method has triggered DDID to achieve higher number of declared duplicates and fewer errors as compared to DuDe. The results of the statistical test show that there is a significant improvement made by DDID in duplicate detection speed even though the data sets are incomplete.

Acknowledgements
We would like to thank the Hasso Plattner Institute (HPI) and Humburg university for making it possible to use data sets and the Duplicate Detection Tool (DuDe) in this research.

Authors' contributions
The corresponding author contributes the concepts and algorithm development. All authors also contribute to content improvement before the final manuscript is submitted. All authors read and approved the final manuscript.

Funding
This research is not funded by any organization.

Availability of data and materials
The data sets used in this research available under these websites https://hpi.de/en/naumann/projects/data-integration-data-quality-and-data-cleansing/dude.html and https://vsis-www.informatik.uni-hamburg.de/oldServer/teaching//projects/QloUD/DaPo/testdata/. A copy of the data sets (after injecting the missing values) used in this study can be obtained through communication with the author.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
All authors agree to have their work published in Big Data Journal.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Faculty of Computer and Information Technology, Aden University, Aden, Yemen. [2]Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Ayer Keroh, Melaka 76100, Malaysia.

Received: 1 April 2021 Accepted: 8 August 2021
Published online: 21 August 2021

References
1. Griffeth RW, Hom PW, Gaertner S. A meta-analysis of antecedents and correlates of employee turnover: update, moderator tests, and research implications for the next millennium. J Manag. 2000;26(3):463–88.
2. Shilane P, Chitloor R, Jonnala UK. 99 deduplication problems. In: 8th USENIX workshop on hot topics in storage and file systems (HotStorage 16), USENIX association, Denver, CO. 2016. p. 1–5.
3. Xia W, Jiang H, Feng D, Douglis F, Shilane P, Hua Y, Fu M, Zhang Y, Zhou Y. A comprehensive study of the past, present, and future of data deduplication. Proc IEEE. 2016;104(9):1681–710.
4. Chernov I, Ivashko E, Rumiantsev A, Ponomarev V, Shabaev A. Survey on deduplication techniques in flash-based storage. In: 2018 22nd conference of open innovations association (FRUCT). IEEE, Jyvaskyla, Finland. 2018.
5. Xu L, Pavlo A, Sengupta S, Ganger GR. Online deduplication for databases. In: proceedings of the 2017 ACM international conference on management of data. ACM, Chicago Illinois USA. 2017; p. 1355–68.
6. Wandhekar V. Validation of deduplication in data using similarity measure. Int J Comput Appl. 2015;116(21):18–22.
7. Menestrina D, Whang SE, Garcia-molina H. Evaluating entity resolution results. In: proceedings of the VLDB endowment. VLDB Endowment, Singapore. 2010;3:208–19
8. Panse F. Duplicate detection in probabilistic relational databases. University of Hambur, PhD thesis. 2015.
9. Umathe VH, Chaudhary G. A review on incomplete data and clustering. Int J Comput Sci Inf Technol. 2015;6(2):1225–7.
10. Wang S, Li M, Hu N, Zhu E, Hu J, Liu X, Yin J. K-means clustering with incomplete data. IEEE Access. 2019;7:69162–71.
11. Subramaniyaswamy V, Pandian C. A complete survey of duplicate record detection using data mining techniques. Inf Technol J. 2012;11(8):941–5.
12. Sadinle M. Detecting duplicates in a homicide registry using a Bayesian partitioning approach. Ann Appl Stat. 2014;8(4):2404–34.
13. Chen Q, Zobel J, Zhang X, Verspoor K. Supervised learning for detection of duplicates in genomic sequence databases. PLoS ONE. 2016;11(8):1–20.
14. Huang Y, Chiang F. Refining duplicate detection for improved data quality. In: TDDL/MDQual/Futurity@ TPDL. 2017.

15. Ali A, Emran NA, Asmai SA, Thabet A. Duplicates detection within incomplete data sets using blocking and dynamic sorting key methods. Int J Adv Comput Sci Appl. 2018;9(9).
16. Wubetie HT. Missing data management and statistical measurement of socio-economic status: application of big data. J Big Data. 2017;4(1):47.
17. Lazar A, Jin L, Spurlock CA, Wu K, Alex S. Data quality challenges with missing values and mixed types in joint sequence analysis. In: data quality challenges with missing values and mixed types in joint sequence analysis. In: 2017 IEEE international conference on big data (Big Data). Boston, MA, USA: IEEE. 2017; p. 2620–7.
18. Elmagarmid AK, Ipeirotis PG, Verykios VS. Duplicate record detection: a survey. IEEE Trans Knowl Data Eng. 2007;19(1):1–16.
19. Monge AE, Elkan CP. An efficient domain-independent algorithm for detecting approximately duplicate database records. In: DMKD; 1997.
20. Bilenko M, Mooney RJ. Learning to combine trained distance metrics for duplicate detection in databases. Technical report, Department of Computer Sciences University of Texas at Austin. 2002.
21. Chen L, Tang C, Yang J, Gao Y. A multilevel and domain-independent duplicate detection model for scientific database. In: Tang C, Yang J, Chen L, Gao Y, editors. Web-Age Information Management. Berlin: Springer; 2010. p. 729–41.
22. Naumann F, Herschel M. An introduction to duplicate detection. Synth Lect Data Manag. 2010;2(1):1–87.
23. Tamilselvi J, Saravanan V. Detection and elimination of duplicate data using token-based method for a data warehouse: a clustering based approach. Int J Dyn Fluids. 2009;5(2):145–64.
24. Köpcke H, Rahm E. Frameworks for entity matching: a comparison. Data Knowl Eng. 2010;69(2):197–210.
25. Alrehamy H, Walker C. SemLinker: automating big data integration for casual users. J Big Data. 2018;5(1):14.
26. Konstantinou N, Abel E, Bellomarini L, Bogatu A, Civili C, Irfanie E, Koehler M, Mazilu L, Sallinger E, Fernandes AAA, Gottlob G, Keane JA, Paton NW. VADA: an architecture for end user informed data preparation. J Big Data. 2019;6(74):1–32.
27. Haque S, Mengersen K, Stern S. Assessing the accuracy of record linkages with Markov chain based Monte Carlo simulation approach. J Big Data. 2021;8(8):1–25.
28. Lehti P, Fankhauser P. Unsupervised duplicate detection using sample non-duplicates. In: Spaccapietra S, editor. Journal on data semantics VII. Berlin, Heidelberg: Springer; 2006. p. 136–64.
29. Bronselaer A, Van Britsom D, De Tré G. Propagation of data fusion. IEEE Trans Knowl Data Eng. 2015;27(5):1330–43.
30. Bharathi B, Reddy CS. Duplicate record deletion in relational database management systems. Int J Sci Eng Res. 2017;8(5):266–71.
31. Babu SA. Duplicate record detection and replacement within a relational database. Adv Comput Sci Technol. 2017;10(6):1893–901.
32. van Gennip Y, Hunter B, Ma A, Moyer D, de Vera R, Bertozzi LA. Unsupervised record matching with noisy and incomplete data. Int J Data Sci Anal. 2018;6(2):1–21.
33. Sitaram D, Dalwani A, Narang A, Das M, Auradkar P. A measure of similarity of time series containing missing data using the mahalanobis distance. In: advances in computing and communication engineering (ICACCE). 2015 second international conference. Dehradun, India: IEEE. 2015; p. 622–7.
34. Abdallah L, Shimshoni I. A distance function for data with missing values and its application. Int J Comput Sci Eng. 2013; p. 7.
35. Emran NA. Data completeness measures. In: Abraham A, Muda AK, Choo Y-H, editors. Pattern analysis. Intelligent security and the internet of things. Cham: Springer International Publishing; 2015. p. 117–30.
36. Emran NA, Embury SM, Missier P. Model-driven component generation for families of completeness. In: QDB/MUD, CTIT workshop proceedings series, Auckland, New Zealand. 2008; p. 123–32.
37. Horton NJ, Kleinman KP. Much ado about nothing: a comparison of missing data methods and software to fit incomplete data regression models. Am Stat. 2007;61(1):79–90.
38. Pigott TD. A review of methods for missing data. Educ Res Eval. 2001;7(4):353–83.
39. Ng SK, Krishnan T, Mclachlan GJ. The EM algorithm. In: James EG, Karl HW, Yuichi M, editors. Handbook of computational statistics: concepts and methods. Berlin: Springer; 2004. p. 137–68.
40. Draisbach U, Naumann F. DuDe: the duplicate detection toolkit. In: proceedings of the international workshop on quality in databases (QDB), Singapore. 2010;10000:1000000.
41. Ellis, B. A consolidated, macro for iterative hot deck imputation. document présenté au NorthEast SAS Users Group-2007. 2007.
42. Song Q, Shepperd M, Chen X, Liu J. Can k-NN imputation improve the performance of C4.5 with small software project data sets? A comparative evaluation. J Syst Softw. 2008;81(12):2361–70.
43. Sim J, Lee JS, Kwon O. Missing values and optimal selection of an imputation method and classification algorithm to improve the accuracy of ubiquitous computing applications. Mathematical Problems in Engineering. 2015. p. 1–14.
44. Bilenko M, Mooney RJ. On evaluation and training-set construction for duplicate detection. In: Proceedings of the KDD-2003 workshop on data cleaning, record linkage, and object consolidation. ACM, Washington, DC. 2003; p. 7–12.
45. Ong S, Pei A. A comparative study of record matching algorithms. PhD thesis, University of Edinburgh, Scotland. 2008.
46. Ektefa M, Marzanah AJ, Sidi F, Memar S, Ibrahim H, Ramali A. A threshold-based similarity measure for duplicate detection. In: IEEE Conference on open systems, IEEE, Langkawi, Malaysia. 2011; p. 37–41.
47. Daggupati B. Unsupervised duplicate detection (UDD) Of query results from multiple web databases. PhD thesis, California State University Channel Islands. 2011.
48. Leitao L, Calado P, Herschel M. Efficient and effective duplicate detection in hierarchical data. IEEE Trans Knowl Data Eng. 2013;25(5):1028–41.
49. Skandar A, Rehman M, Anjum M. An efficient duplication record detection algorithm for data cleansing. Int J Comput Appl. 2015;127(6):28–37.

50. Bo C, Wang K, Fox JJ, Skadron K. Entity resolution acceleration using the automata processor. In: proceedings—2016 IEEE international conference on big data, Big Data. 2016; p. 311–8.
51. Priyanka M, Baby A. A survey on various duplicate detection methods. Int J Comput Sci Inf Technol. 2017;8(1):7–9.
52. Meshram MT. Duplicate detection with map reduce and deletion procedure. Int J Comput Trends Technol. 2017;48(2):51–3.
53. Zieger T. Self-adaptive data quality automating duplicate detection. PhD thesis, Potsdam. 2018.
54. Hildebrandt K, Panse F, Wilcke N, Ritter N. Large-scale data pollution with apache spark. IEEE Trans Big Data. 2020;6(2):396–411.
55. Yan S, Lee D, Kan M-Y, Giles LC. Adaptive sorted neighborhood methods for efficient record linkage. In: proceedings of the 7th ACM/IEEE-CS joint conference on digital libraries. ACM, Vancouver BC Canada. 2007; p. 185–94.
56. Hernández MA, Stolfo SJ. The merge/purge problem for large databases. ACM SIGMOD Rec. 1995;24:127–38.
57. Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady. 1966;10(8).

Publisher's Note