Journal of Mathematics and Computing Science, 12 (1): 12-22, 2026
Copyright © UiTM Press
eISSN: 0128-0767

Experimental Analysis of the Impact of Matching String Length on Duplicate Detection Accuracy

Abdulrazzak Ali[1*]

[1]Faculty of Computer and Information Technology, University of Aden, Aden, Yemen.

Authors' email: dr.abdullrazak.fcit@aden-univ.net[*]

[*]Corresponding author

Received 9 March 2026; Received in revised 4 May 2026; Accepted 25 May 2026
Available online 15 June 2026
DOI: https://doi.org/10.24191/7t9vdw78

Abstract: String matching is a fundamental problem in many computer applications, such as information retrieval, natural language processing, and database duplicate detection. A key issue in this context is the impact of string length on the accuracy and efficiency of matching algorithms, as variations in length can significantly affect the ability to distinguish between similar or identical records. This paper presents a systematic experimental analysis of the effect of string length on the accuracy of duplicate record detection. To achieve this, experiments were conducted using the standardized CD dataset of 9,763 real audio CD records, a benchmark commonly used to evaluate duplicate detection techniques. The evaluation was performed using the Duplicate Detection within Incomplete Datasets (DDID) model across several different matching string lengths. By analyzing detection accuracy and the resulting error rates, the experimental results demonstrate a clear relationship between string length and matching performance, revealing a distinct trade-off between false positives and false negatives. These findings highlight that selecting an appropriate string length is a crucial factor in improving the accuracy and reliability of duplicate detection methods

Keywords: Duplicate Detection, Clustering Method, Matching String, Similarity Threshold.

1 Introduction

The rapid proliferation of digital technologies, connected devices, and online services has resulted in unprecedented growth in the global volume of data. According to estimates based on International Data Corporation (IDC)'s Global Datasphere, the amount of data generated worldwide was approximately 33 zettabytes (ZB) in 2018 and was projected to increase to 175 ZB by 2025 [1]. This exponential growth in digital data has not only increased the need for efficient storage and processing but has also heightened the importance of data quality and management across a wide range of organizations and applications. As datasets expand in volume and complexity, errors and inconsistencies in stored records become more difficult to identify and resolve. One of the most common data quality challenges is the presence of duplicate records, which often arise from repeated data entry, variations in record representation, changes in data sources, or the integration of information from multiple systems [2], [3]. Duplicate records can adversely impact data analysis and decision-making while increasing the computational cost of downstream data processing tasks [4]. Consequently, reliable duplicate detection is a critical component of data cleaning and quality management.

This challenge is particularly evident in large-scale online platforms, where data are continuously generated, collected, and integrated from diverse sources. For example, online retail platforms often maintain extensive and continuously growing product catalogues compiled from multiple suppliers.

This is an open access article under the CC BY-SA license (https://creativecommons.org/licenses/by-sa/3.0/).

Variations in product descriptions and modifications introduced by different data providers can result in duplicate records [5]. Similarly, online applications and social media platforms generate large volumes of data that may contain substantial redundancy and duplicate records [6]. As the volume and diversity of such data continue to increase, accurately and efficiently detecting duplicate records becomes increasingly challenging. This highlights the need for robust duplicate detection techniques capable of operating effectively on large and heterogeneous datasets.

Duplicate detection within incomplete datasets is a complex task because missing data can make records appear different even when they refer to the same real-world entity. The process of detecting duplicates within incomplete datasets is particularly challenging in terms of the following aspects:

I. Clustering Data: In duplicate detection, datasets are often partitioned into smaller groups or clusters to reduce the computational cost of pairwise comparisons. However, clustering incomplete datasets can be challenging, particularly when the attributes used to construct a sorting key contain missing values. Missing values may reduce the probability that records referring to the same real-world entity will be assigned to the same group. Therefore, it is necessary to identify appropriate attributes for constructing the sorting key based on their uniqueness and completeness properties. Appropriate attributes are those selected based on a low rate of value repetition and a low rate of missing values, respectively. However, selecting attributes solely based on uniqueness and completeness may be insufficient if the selected attributes contain missing values, particularly when the attribute forming the first part of the sorting key is missing. Therefore, compensating for missing values in the selected attributes is necessary to increase the probability that records referring to the same real-world entity will be assigned to the same group [7].
II. Data Matching: This process is important for identifying duplicate records within a dataset. The matching process involves comparing attribute values using different approaches, such as character-to-character comparison, token-to-token comparison, or phonetic comparison. However, standard matching methods may perform poorly when the datasets being matched contain incomplete records. Missing values can cause similar records to appear different, thereby reducing the accuracy of the matching process. Furthermore, comparing all attribute values across records can be computationally expensive, particularly when dealing with large datasets. Therefore, selecting suitable strings or tokens from the attributes that increase the probability of correctly matching records is important. The creation of comparison strings using the same criteria as those applied to select attributes for constructing sorting keys may therefore be a logical approach.

In fact, the problem of detecting duplicate records becomes more complex when no universal key is available or when unique identifiers are missing or unreliable within the record set [8], [9], [10]. During the duplicate detection process, matching candidate record pairs in incomplete datasets involves a trade-off between false-positive and false-negative errors, which can be influenced by the length of the selected matching string. Dynamic matching strings can improve this trade-off by enabling more flexible comparisons and potentially achieving higher matching accuracy. This approach also makes the trade-off easier for users to define because the matching tokens are dynamically selected from attributes that receive higher ranks based on their relevance to the matching process.

The proposed approach is particularly useful in the following situations:

1. A user may have limited time to perform data cleansing before applying the duplicate detection algorithm. The proposed approach can reduce the need for extensive preprocessing, thereby allowing the algorithm to be applied efficiently while maximizing the number of duplicate records identified.
2. A user may have limited knowledge of the characteristics of the given dataset but still needs to identify high-ranking attributes that meet specific criteria, such as a high proportion of distinct values and a low proportion of missing values.
3. A user working with an incomplete dataset may need to compensate for missing values using a compensation algorithm to improve the matching rate.

4. A user may need to partition a dataset into smaller windows to reduce the number of pairwise comparisons. In this case, the sorting key and matching token string can be generated dynamically based on the characteristics of the dataset.
5. A user may need to determine an appropriate matching token string length. The algorithm can be executed multiple times using different string lengths to identify a suitable length and reduce the trade-off between false-positive and false-negative errors.
6. A user may use result curves to evaluate duplicate detection performance on incomplete datasets. These curves can illustrate the number of detected duplicates relative to the actual number of duplicates in the dataset and demonstrate the trade-off resulting from changes in the matching string length.

In this paper, we analyzed the data matching process as one of the key stages in duplicate detection and investigate the factors that influence matching accuracy in incomplete datasets. In particular, we examined the trade-off associated with varying the matching string length using the proposed Duplicate Detection within Incomplete Data (DDID) model, which is designed to operate on both small and large datasets. The DDID model performs data matching using dynamically generated matching strings derived from high-ranking attributes, providing a more effective and efficient approach to matching records in incomplete datasets. The proposed model is intended to improve the efficiency of duplicate detection in incomplete datasets compared with traditional duplicate detection methods [7].

2 Background

String matching in computer science refers to the process of comparing two strings to determine whether they are identical or sufficiently similar according to a specified criterion. It can also be defined as a method for analyzing text strings to identify specific patterns within them [11]. In contrast to pattern recognition, string matching may involve either exact or inexact matching based on a predefined similarity threshold. The comparison process uses various similarity measures to determine the degree of similarity between strings. Despite undergoing data preprocessing, grammatical and semantic errors may still occur in the data [9]. Therefore, several techniques have been developed to address these issues, with each technique being designed to handle specific types of errors. Common string matching techniques include q-grams, edit distance, and the Jaro distance metric [12]. The main objective of string matching algorithms is to improve computational efficiency by reducing the number of comparisons required while maintaining an acceptable level of matching accuracy [13].

The traditional approach to similarity computation is based on testing whether two strings are equivalent. However, this approach is no longer sufficient in the presence of typographical errors and other data quality issues. To improve the quality of data matching, several types of similarity measures have been developed, each designed to handle specific types of input and data errors. Similarity measures for identifying matching strings can be broadly classified into three groups: character-based similarity measures, token-based similarity measures, and numeric similarity measures.

A Character-Based Similarity Measures

This approach compares strings at the character level by measuring the similarity or distance between two text strings. Several similarity measures follow this approach, including the Jaccard coefficient, Q-Gram similarity function, Levenshtein distance, and Jaro similarity. These measures are described in this section to illustrate how character-based similarity is used to determine whether two strings are matching or non-matching. Let S1 and S2 denote two strings used to illustrate the application of these similarity measures in determining their degree of similarity.

- Jaccard coefficient: the general measure to compare between two sets P and Q as follow:

$$Jaccard\ (P, Q) = \frac{|P \cap Q|}{P \cup Q} \quad (1)$$

The Jaccard method measures the similarity between two strings based on the proportion of common elements between them. In this approach, each string (S) is divided into a set of

substrings, represented as {S1, S2, S3..., Sn}. The Jaccard coefficient is then calculated as follows:

$$StringJaccard(S1,S2) = \frac{|tokenize(S1) \cap tokenize(S2)|}{tokenize(S1) \cup tokenize(S2)} \quad (2)$$

- Q-Gram Similarity Function: The Q-Gram similarity function measures the similarity between two strings by dividing each string into overlapping substrings of length (q), known as q-grams. The division of a string into q-grams is generally not dependent on delimiters or special characters such as spaces, commas, or line breaks. The type of q-gram is determined by its length: a one-character substring is called a unigram, a two-character substring is called a bigram, and a three-character substring is called a trigram. Substrings with larger values of (q) are generally referred to as q-grams. The q-grams generated from a string are overlapping, meaning that consecutive q-grams share one or more characters. The use of q-grams is beneficial because two strings that are similar or close to each other in terms of character composition are likely to share a large number of q-grams [14]. The similarity between two strings, S1 and S), can be calculated using the Q-Gram similarity formula, with the resulting similarity value ranging from 0 to 1:

$$sim_{q-gram(S1,S2)} = \frac{1}{2}\left(\frac{|G_{S1} \cap G_{S2}|}{|G_{S1}|} + \frac{|G_{S1} \cup G_{S2}|}{|G_{S2}|}\right) \quad (3)$$

Where |GS1∩GS2| is the number of common q-grams between S1 and S2, |GS1|and |GS2| is the number of q-grams of S1 and S2, respectively.

- Levenshtein Distance: The Levenshtein distance represents a set of editing procedures required to enter, replace, and delete characters in the S1 series to convert them into S2. The algorithm, which is based on the dynamic programming, is from the best algorithms for the calculation of the Levensthein between the two strings [15]. It starts with the initialization of the matrix, M by (|S1|+1) x (|S2|+1), where |S| represents the length of the string S. During the process, the values of the matrix elements are computed according to the formulas adopted in continuity. This is symbolized by matrix $M_{i,j}$ where i represents the column and j represents the row which is being marked by $M_{i,j}$ where ($0 \leq i \leq |S1|i, 0 \leq j \leq |S2|$). The matrix M is calculated from the right side to the left to calculate Levenshtein distance [16]. Levenshtein distance is an effective tool to identify typos and spelling errors, but it is not considered as useful for other types of errors [12], [17].

$$M_{i,0} = i\,,\ M_{0,j} = j$$

$$M_{i,j} = \begin{cases} M_{(i-1,j-1)}, & if\ S1 = S2; \\ +\min(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1}) & for\ other\ cases \end{cases}$$

- Jaro Similarity: For Jaro similarity method, there are two types of distance:
  a) Jaro Distance: The Jaro distance measures the similarity between two strings, S1 and S2, based on the number of common characters and their positions within the strings [16]. Two characters are considered common if they are identical and their positions in the two strings do not differ by more than half the length of the shorter string. The positions of the characters in S1 and S2 are represented by i and j, respectively. The steps for calculating the similarity between the two strings can be illustrated using the Jaro algorithm.

$$\frac{max(|S1|,|S2|)}{2} - 1 \quad (4)$$

$$Sim_{jaro} = \frac{1}{3}\left(\frac{m}{S1} + \frac{m}{S2} + \frac{m-t}{t}\right) \quad (5)$$

Where t is the number of transportations

  b) Jaro-Winkler Distance: It is considered an extension of the Jaro distance method. The standard Jaro-Winkler similarity is commonly expressed as:

$$JW = J + (l \times p \times (1 - J)) \quad (6)$$

where:

- $J =$ Jaro similarity
- $l =$ length of the common prefix, usually limited to 4
- $p =$ scaling factor, typically 0.1

Character-based similarity measures compare two records on a character-by-character basis. These approaches may be influenced by the size of the records, which can, in turn, affect the performance of duplicate detection. However, character-based approaches can provide considerable accuracy because all available characters are systematically examined during the comparison process.

i. Token-Based Similarity Measures

These measures compare two strings by partitioning them into a sequence of tokens. These metrics are particularly effective in handling typographical errors, especially in cases where character-level metrics may fail to identify similarities between entities [18].

ii. Numeric Similarity Measures

Several methods have been developed to detect similarities between strings, while methods for identifying similarities in numerical data are relatively less developed. In some approaches, numerical values are treated as strings, allowing numbers with similar values to be identified based on their character representations. Another approach involves extending the concept of cosine similarity to numerical data and evaluating its effectiveness in detecting duplicate records.

Generally, there are relatively few differences among the three approaches in terms of scalability and performance. Clustering-based approaches can provide good performance because they categorize records into groups, thereby reducing the search space and facilitating the identification of duplicate records. However, a key challenge is to develop methods that minimize unnecessary comparisons during the duplicate detection process. Overall, each technique has its own strengths and limitations, depending on the characteristics of the data and the requirements of the duplication detection task.

B Similarity Threshold

In string matching, various similarity measures and threshold values are used to determine whether pairs of strings match. The selection of an appropriate similarity measure depends on the nature of the data and the types of errors that may contribute to duplicate records. Similarly, the similarity threshold depends on the size and characteristics of the dataset. A duplicate detection configuration may perform well on a test dataset but perform poorly when applied to datasets of different sizes. This limitation is partly attributed to the transitive nature of duplication. In larger datasets, transitivity may cause more records to be included in a duplicate cluster than intended. The similarity threshold is therefore used to classify pairs of strings as either duplicates or non-duplicates. If the similarity score between two compared strings is higher than the specified threshold, the record pair is considered a duplicate, and both records are assigned to the same cluster [19].

Draisbach and Naumann [20] highlighted the problems associated with selecting an appropriate threshold value. Figure 1(a) illustrates this threshold problem using three records, A, B, and C, where edges represent similarity scores between records. Records A and B are sufficiently similar – with a similarity value between 0.8 and 0.9 – are thus assigned to the same cluster. Consequently, records A and B are classified as duplicates while pairs (A, C) and (B, C) are classified as non-duplicates. When a new record, D, is inserted into the dataset, record D is identified as a duplicate of record C, as illustrated in Figure 1(b). Therefore, the pair (C, D) is classified as a duplicate, while the pair (A, D) is classified as a non-duplicate. However, a problem arises with the pair (B, D). This pair may be classified as a duplicate if the selected threshold value is ≤ 0.84, even though records B and D are actually non-duplicates. This problem occurs because of the transitive nature of duplicate clustering, whereby a record may become associated with a cluster through its similarity to other records, even when it does not directly satisfy the intended duplicate relationship with every record in that cluster.

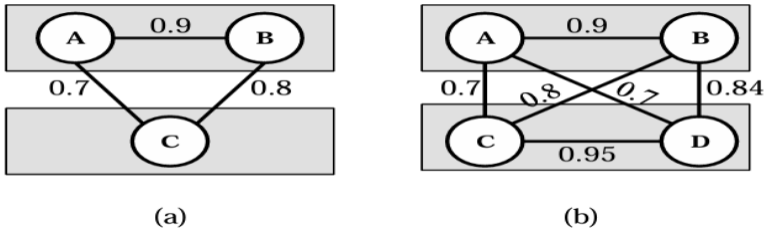

Figure 1: Illustration of threshold selection problem [20]

Accordingly, the appropriate selection of a threshold value plays an important role in accurately classifying pairs of records as duplicates or non-duplicates. The selection of the threshold value may also depend on the size and characteristics of the dataset. In the next section, the DDID model for duplicate detection in incomplete datasets is discussed to illustrate the trade-offs that may arise when analysing duplicate detection results due to variations in the length of the comparison strings.

3 Method

In this section, the trade-off between false negatives and false positives arising from variations in the comparison string length is examined in the context of duplicate detection results.

A DDID Model

To examine the effect of comparison string length on the matching process, an experiment was conducted using the DDID model, which has been applied to detect duplicate records in incomplete datasets [7][21]. The model relies on comparison strings constructed by combining the values of highly ranked attributes to identify duplicate records. The DDID model consists of the following basic phases:

- Attribute selection: Selecting highly ranked attributes that meet the criteria of uniqueness and completeness.
- Data clustering: Dividing the dataset into smaller groups based on dynamic sorting keys to reduce the number of comparisons and improve processing efficiency.
- Similarity calculation: Calculating the similarity between comparison strings using appropriate similarity measures.
- Duplicate classification: Classifying record pairs as duplicates or non-duplicates based on their similarity scores and the specified threshold.

B Data Set

To conduct the experiment, we used the CD dataset provided by the Hasso Plattner Institute (HPI)[1], which has been widely used in duplicate detection research. The CD dataset was extracted from the freeDB repository. It consists of 9,763 records representing real audio CDs, with each record containing information such as the artist, title, and tracks. The dataset contains 299 duplicate records. The CD dataset has undergone a series of modifications, as reported in [7][21].

C Experiment Configuration

All experiments were conducted under the same execution conditions, as follows:

- The Sorted Neighborhood Algorithm was used to pair and sort the records, with the window size set to 20.
- The Levenshtein distance was used to compare the matching strings (CS), with the similarity threshold set to 0.9.

After the matching process, the record pairs were classified as duplicates or non-duplicates. The post-processing phase was configured to compute the transitive closure and generate the final output containing the record pairs classified as duplicates in a JSON file.

In the DDID model, comparison strings are constructed from the selected attributes based on the criteria of uniqueness and completeness of the attribute values. This approach avoids selecting attributes that may result in a high false-positive rate. For example, a comparison string (token) created from attributes such as city or zip code may incorrectly identify candidate record pairs as duplicates when

they are, in fact, non-duplicates. Conversely, attributes with a high proportion of missing values may produce incomplete or distorted comparison strings, which can increase the false-negative rate.

In our experiments, the matching strings were constructed from the values of the first three highest-ranked attributes, which had high distinct-value ratios and low proportions of missing values. In the first experiment, the matching string consisted of nine characters, created by combining the first three characters of each selected attribute. In the second experiment, the matching string consisted of 24 characters, created by combining the first eight characters of each selected attribute. In the third experiment, the matching string consisted of 33 characters, created by combining the first eleven characters of each selected attribute. Listing 1 presents the method used to construct the matching strings.

Listing 1: Comparative Strings algorithm

```
void create_cstr() throws FileNotFoundException{
String[]     str=new     String[values.size()];     String     s;     for(int
i=0;i<values.size();i++) {
 str[i] = null; int temp=0; String strcom;
 for(int j=1;j<4;j++){
         temp=rep[i][j].length();
         strcom=rep[i][j].replaceAll("\\s+","");
          if( str[i]==null && (strcom!=null ||strcom!=" "))   {
                 if(temp>=3)
                        str[i]=strcom.substring(0,3);
                 else if(temp>=2)
                         str[i]=strcom.substring(0,2);
                 else if(temp>=1)
                         str[i]=strcom.substring(0,1);   }
           else if((strcom!=null ||strcom!=" ") && str[i]!=null)   {
                        if(temp>=3)
                                     str[i]+=strcom.substring(0,3);
                         else if(temp>=2)
                                       str[i]+=strcom.substring(0,2);
                         else if(temp>=1)
                                       str[i]+=strcom.substring(0,1);}
                                  else    {
                                             str[i]+=""; }}
       System.out.println(str[i]); }}
```

Based on the experiments conducted on the CD dataset to identify duplicate records, three attributes – title, artist, and cdextra – were selected to construct the matching strings. Figure 2 presents a portion of the CD dataset and the matching strings generated using the first three characters of each selected attribute, as indicated in the CS attribute. The results show that some of the matching strings were not significantly affected by missing values in the selected attributes. This is because the DDID model incorporates an algorithm that compensates for missing values when corresponding values are available in other records.

CS	[PK	artist	title	category	genre	cdextra	year	year1]
zygvvy	[100597	vvysotskij	zygankaskartami	folk	folk		1999	1998]
zwngnt	[100804	gntalaras	zwntaneshxografhseis	folk	greek		2004	null]
zwivar	[102180	various	zwischentagundtraumvol4cd	misc	dream		null	1993]
zurohl	[106774	ohl	zurckzurfront	rock	punk		2003	id3g:17]
zurmar	[101052	marsvenus	zurka	jazz	dance		null	2001]
zunnor199	[102328	nordwind	zuneuenufern	misc	vocal	1999	1999	2001]
zumrogid3	[101509	rogerwhittaker	zumweinenistimmernochzeit	misc	easylisteni	id3g98	null	null]
zuhtrumet	[107599	truckstop	zuhause	country	country	metronom	1977	1902]
zuhdie	[103156	dieblauenjungs	zuhausezuhause	misc	oldies		null	null]
zugwol196	[103576	wolfbiermann	zugastbeiwolfgangneuss	misc	satire	1965	1965	1997]
zrazra	[102299	zramalho	zramalho-parcerias	misc	null		null	null]
zoou2199	[105619	u2	zootopia-hottestticketinrock	rock	rock	1993	1993	2002]
zoou2	[107404	u2	zootvtour-disc1	rock	null		null	2003]
zoothe	[105368	theknack	zoom	rock	null		null	1998]
zodmar	[104824	marillion	zodiac	rock	null		null	1993]
zoblin	[107616	lintvern	zobcasa	country	pop		1999	null]
zmar	[108275	marceyhamm	z	newage	newage		null	1996]
zlovar	[100124	variousartists	zloteprzebojepolskiegosocjal	folk	folk		1999	null]
zlopid	[103955	pidzamaporno	zlodziejezapalniczek	misc	null		null	1999]

Figure 2: CD data set with matching strings

According to the results of the experiments, there is a discrepancy between the values of the types of errors in the three experiments, where the ratio of false-positive and false-negative decreased with a higher ratio of true positives in the third experiment than in the first and second experiment. Table 1 shows the results of the three experiments.

Table 1: Duplicate Detection Results for Different Matching String Lengths

True Duplicates	String Length	Detected Duplicates	True Positives	False Negatives	False Positives
299	9	2956	213	86	2743
299	24	252	176	123	76
299	33	281	240	59	41

According to the results presented in Table 1, the false-positive rate was higher in the first experiment, which used a nine-character matching string comprising three characters from each selected attribute. This may be attributed to the increased likelihood of similarity among the initial characters of the selected attributes. For example, if the values of a selected attribute begin with common terms such as “the” or “http,” the matching process may incorrectly identify non-duplicate records as duplicates, resulting in a higher false-positive rate. In the second experiment, which used a 24-character matching string, the false-positive rate decreased, while the false-negative rate increased. Finally, the third experiment demonstrated a marked improvement, with both the false-positive and false-negative rates decreasing when a 33-character matching string was used. These findings indicate that increasing the length of the matching string can improve the accuracy of duplicate detection. Figure 3 illustrates the trade-off associated with the length of the matching string.

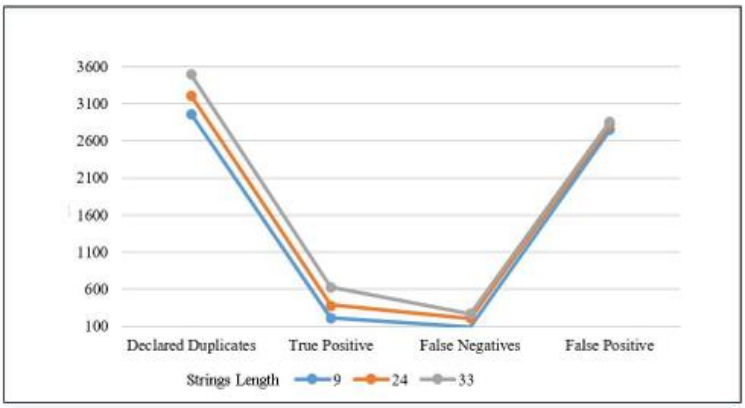

Figure 3:Duplicates Detection Error Trade-off

The improvement in the results of the third experiment is due to the proportion of data matching between the attribute values, where some attributes contain values that may appear somewhat similar with a slight difference toward the end of the strings. Table 2 and Figure 4 illustrate the efficiency and effectiveness of duplicate detection in a CD dataset using different matching string lengths in each test.

Table 2:Accuracy of DDID Experiments

Experiment	Recall	Precision	F-Score	Accuracy
Experiment 1	0.71	0.072	0.12	0.71
Experiment 2	0.588	0.698	0.637	0.979
Experiment 3	0.80	0.85	0.82	0.99

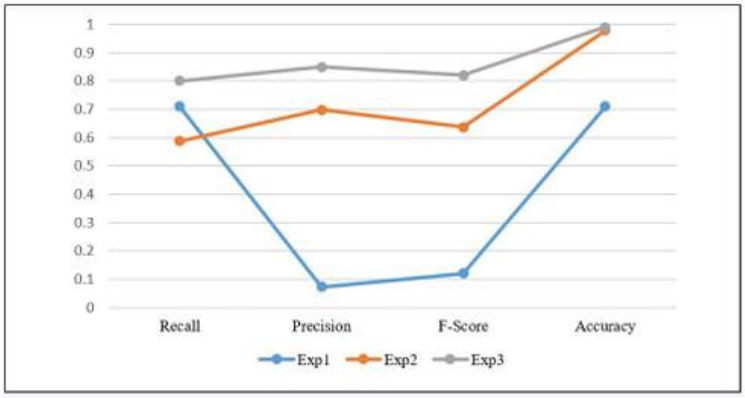

Figure 4:The Accuracy of Matching strings for Detecting Duplicates

Finally, it is important to consider the data size and data type when determining the appropriate length of the matching string. Constructing a matching string using the complete values of the selected attributes without specifying a fixed length may increase execution time due to the larger number of characters involved in the comparison, particularly when the selected attributes contain long text values. On the other hand, specifying a fixed length for the matching string may, in some cases, result in false positives when the values of the selected attribute or attributes differ only in their later or final characters. Therefore, the length of the matching string should be carefully determined based on the characteristics of the dataset and the types of attributes being compared.

Based on these observations, an automated approach for estimating the number of characters to be extracted from each selected attribute, based on the characteristics of the dataset, may provide a

promising direction for improving duplicate detection. Such an approach could consider factors such as attribute uniqueness, the proportion of missing values, and the observed similarity distributions to determine an appropriate matching string length. Developing and evaluating a formal automated method for estimating the optimal matching string length represents an important area for future research.

4 Conclusion

This paper demonstrates that the effectiveness of string matching in duplicate detection depends largely on the characteristics of the data used to construct the matching strings. The construction of matching strings is also influenced by the selection of appropriate attributes, which can help reduce the number of comparisons, improve processing efficiency, and increase the accuracy of duplicate detection.

The experimental results demonstrate a trade-off between false-positive and false-negative errors resulting from the selection of the matching string length. The length of the matching string can significantly influence the accuracy of duplicate detection. In particular, the results indicate that an appropriate matching string length can reduce both types of errors and improve the overall effectiveness of the duplicate detection process.

This study has a limitation in that the experimental evaluation was conducted using a single audio CD dataset. Although the results demonstrate a clear relationship between matching string length and duplicate detection performance, the generalizability of this relationship to other types of datasets remains to be investigated. Future research should evaluate the proposed approach using additional datasets with different characteristics, such as bibliographic, census, and other real-world datasets. Such evaluation would help determine whether the observed impact of matching string length remains consistent across different data domains, structures, and attribute characteristics.

Conflict of Interest Statement

The author declares no conflict of interest related to this research, authorship, and publication of this manuscript.

Author Contributions

The author carried out all aspects of this research, including study design, methodology development, data analysis, experimentation, and manuscript preparation.

References

[1] M. Stankovic, A. A. Garba, and N. Neftenov, Emerging Technology Trends: Artificial Intelligence and Big Data for Development 4.0. Geneva, Switzerland: International Telecommunication Union (ITU), 2021. [Online]. Available: https://www.itu.int/pub/D-TND-02-2021.

[2] X. Chu and I. F. Ilyas, “Qualitative data cleaning,” Proc. VLDB Endow., vol. 9, no. 13, pp. 1605–1608, 2016, doi: 10.14778/3007263.3007320.

[3] I. F. Ilyas and X. Chu, Data Cleaning. New York, NY, USA: Association for Computing Machinery (ACM), 2019, doi: 10.1145/3310205.

[4] T. Papenbrock, A. Heise, and F. Naumann, “Progressive duplicate detection,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 5, pp. 1316–1329, May 2015, doi: 10.1109/TKDE.2014.2359666.

[5] O. S. Albayrak, T. Aytekin, and T. A. Kalaycı, “Duplicate product record detection engine for e-commerce platforms,” Expert Syst. Appl., vol. 193, Art. no. 116420, May 2022, doi: 10.1016/j.eswa.2021.116420.

[6] Y. Mu, M. Jin, X. Song, and N. Aletras, “Enhancing data quality through simple de-duplication: Navigating responsible computational social science research,” in Proc. 2024 Conf. Empirical Methods in Natural Language Processing (EMNLP), 2024, pp. 12477–12492, doi: 10.18653/v1/2024.emnlp-main.694.

[7] A. Ali et al., “Duplicates detection within incomplete data sets using blocking and dynamic sorting key methods,” Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 9, pp. 629–636, 2018, doi: 10.14569/IJACSA.2018.090979.

[8] P. Christen, Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-31164-2.
[9] F. Panse, "Duplicate detection in probabilistic relational databases," 2015. [Online]. Available: https://ediss.sub.uni-hamburg.de/volltexte/2015/7430/. [no doi] Panse, F., Duplicate Detection in Probabilistic Relational Databases. Ph.D. dissertation, University of Hamburg, Hamburg, Germany, 2015. [Online]. Available: https://ediss.sub.uni-hamburg.de/bitstream/ediss/5911/1/Dissertation.pdf
[10] S. Aleshin-Guendel and R. C. Steorts, "Convergence diagnostics for entity resolution," Annu. Rev. Stat. Appl., vol. 11, pp. 419–435, Apr. 2024, doi: 10.1146/annurev-statistics-040522-114848.
[11] R. Ramos-Frías and M. Vargas-Lombardo, "A review of string matching algorithms and recent implementations using GPU," Int. J. Secur. Appl., vol. 11, no. 6, pp. 69–76, Jun. 2017, doi: 10.14257/IJSIA.2017.11.6.06.
[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," IEEE Trans. Knowl. Data Eng., vol. 19, no. 1, pp. 1–16, Jan. 2007, doi: 10.1109/TKDE.2007.250581.
[13] D. Gurung, U. K. Chakraborty, and P. Sharma, "Intelligent predictive string search algorithm," Procedia Comput. Sci., vol. 79, pp. 161–169, 2016, doi: 10.1016/j.procs.2016.03.116.
[14] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," Theor. Comput. Sci., vol. 92, no. 1, pp. 191–211, Jan. 1992, doi: 10.1016/0304-3975(92)90143-4.
[15] N. Gali, R. Mariescu-Istodor, and P. Fränti, "Similarity measures for title matching," in Proc. 23rd Int. Conf. Pattern Recognition (ICPR), 2016, pp. 1548–1553, doi: 10.1109/ICPR.2016.7899857.
[16] D. Hadzic, N. Sarajlic, and J. Malkic, "Different similarity measures to identify duplicate records in relational databases," in Proc. 24th Telecommunications Forum (TELFOR), 2016, doi: 10.1109/TELFOR.2016.7818899.
[17] V. Wandhekar and A. Mohanpurkar, "Validation of deduplication in data using similarity measure," Int. J. Comput. Appl., vol. 116, no. 21, pp. 18–22, Apr. 2015, doi: 10.5120/20460-2819.
[18] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," Int. J. Comput. Appl., vol. 68, no. 13, pp. 13–18, Apr. 2013, doi: 10.5120/11638-7118.
[19] M. M. Amin, D. Stiawan, Ermatita, and R. Budiarto, "Proposed threshold-based and rule-based approaches to detecting duplicates in bibliographic database," Bull. Electr. Eng. Informatics, vol. 13, no. 3, pp. 2036–2047, Jun. 2024, doi: 10.11591/eei.v13i3.7665.
[20] U. Draisbach and F. Naumann, "On choosing thresholds for duplicate detection," in Proc. 18th Int. Conf. Information Quality (ICIQ), 2013. [Online]. Available: https://hpi.de/oldsite/fileadmin/user_upload/fachgebiete/naumann/publications/PDFs/2013_draisbach_on.pdf. [no doi]
[21] A. Ali, N. A. Emran, and S. A. Asmai, "Missing values compensation in duplicates detection using hot deck method," J. Big Data, vol. 8, Art. no. 112, 2021, doi: 10.1186/s40537-021-00502-1.