International Journal of Service and Knowledge Management
International Institute of Applied Informatics
2025, Vol. 9, No. 1, IJSKM891

A Study of NLPDedup Efficiency for Small and Large Datasets

Hinata Yokoyama [*], Kazuma Iwamoto [*],
Ichitoshi Takehara [*], Kazuaki Ando [*], Hitoshi Kamei [*]

Abstract

Recently, the amount of data has grown rapidly. The deduplication functions reduce the amount of data by finding and deleting the redundant data. Meanwhile, to check data redundancy, the functions affect the performance because they issue many read I/Os and compare data. To mitigate the performance penalty, it is effective to narrow down processed files. Conventional methods use file metadata and hash values generated from file contents as indicators. However, if many files are stored in a file system, the methods are not efficient because of high load caused by checking metadata and hash value calculation. We propose a novel method, called NLPDedup, to narrow down files by using natural language processing for data deduplication functions. NLPDedup uses file names as indicators for narrowing down target files. This paper describes the overview of NLPDedup, how NLPDedup determines the target files, and the evaluation results of small and large datasets. From the results, the threshold of NLPDedup indicators needs to be set in terms of the natural language processing algorithms and the datasets. Consequently, we found that NLPDedup is effective in both datasets, and it is more effective by setting appropriate thresholds.

Keywords: Data deduplication, Natural Language Processing, Levenshtein distance, N-gram

1 Introduction

Recently, the amount of important data has grown as the generated data increases. The important data needs to be backed up. To reduce the amount of backup data, some methods are proposed. One of the methods is the compression function. The compression function encodes target data to smaller-encoded data. Generally, LZ compression is used for compression function. Meanwhile, delta compression is normally used for backup storage systems [1][2][3][4]. The delta compression finds similar data chunks and stores the differential region between these data [5][6]. On the other hand, the other method is data deduplication functions [7][8][9][10]. Note that the data deduplication can be combined with the delta compression. These functions are mainly implemented in the backup storage systems.

[*] Kagawa University, Kagawa, Japan

The deduplication function finds the redundant data in a file system, and it deletes them to reduce the amount of data. There are two types of deduplication functions. One is whole-file level, the other is file-block level [11]. Whole-file level deduplication targets all the data of a file at once. For instance, if the contents of file A and file B are completely same, then whole-file level deduplication can delete the redundant data. Meanwhile, file-block level finds duplicated data in a part of file. If a part of contents of file A and file B is same, then file-block level makes the data of the contents single.

Both types of data deduplication functions have performance problems. To find the redundant data, the deduplication function reads the files in a filesystem. When many files are stored in the filesystem, likewise a large amount of data, it issues many read I/O to find the data. Thus, the I/O load becomes high. The one of the solutions for this issue is filtering to narrow down target files. Conventional methods use file metadata, such as file size and extension, and hash values generated from file contents as indicators. For instance, when whole-file level deduplication is applied, the files with different file sizes are eliminated from the target files. However, each indicator is insufficient because the number of files grows dramatically and the amount of data increases exponentially.

Therefore, we propose a novel method, called Natural Language Processing for Data Deduplication (NLPDedup) [12], for narrowing down target files by using file names. We assume that a file name summarizes the file content; thus, if the file name is similar, then the file contents are also similar. For instance, if the names of two files are produced by a government office "financial report", then these files contain the table data to show balance. Thus, the data may be redundant, and it can be reduced.

This paper shows the system overview of NLPDedup, the calculation formula of threshold determination, and the evaluation results. NLPDedup consists of two modules, one is deduplication judgement module, the other is deduplication module. NLPDedup adds a new step using file names to the conventional methods for deduplication judgement module. NLPDedup applies the conventional methods to narrow down target files in advance. The evaluation results of our hypothesis and the equation to calculate the similarity threshold are described. From the evaluation results, we conclude that NLPDedup can be effectively applied for both small and large datasets by setting appropriate thresholds.

2 Background

2.1 Data Deduplication

Data deduplication is a function for data reduction. The function finds the redundant data in a filesystem and deletes it. To find it, the function reads the data stored in the filesystems and compares them. There are two types of deduplication functions, one is whole-file level, the other is file-block level. Moreover, the file-block level is divided into two methods in terms of block size to be processed. The overview of these types is as follows, and we focus on (2) fixed block-file and (3) variable length block-file.

(1) Whole-file: Whole-file deduplication is performed on file-by-file. This method achieves 30% ~ 60% of the deduplication rate [11].

(2) Fixed block-file: Fixed block-file deduplication splits a file into fixed-length blocks, and it compares each block. This method achieves 40% ~ 60% of the deduplication rate [11].

(3) Variable length block-file: Variable length block-file deduplication splits a file into variable-length blocks, and it compares each block. This method achieves 50% ~ 80% of the deduplication rate [11].

2.2 Conventional Filtering Methods

Data deduplication functions issue many read I/Os to find the redundant data. To mitigate the I/O performance penalty, the function needs to avoid ineffective processes for data deduplication. Therefore, target files must be narrowed down by filtering. Conventional filtering methods are as follows:

- *file size*: This method filters target files by file size. For example, if there are two files with different sizes, the data of these files may be different. Especially, whole-file deduplication cannot deduplicate the different file size.
- *extension:* This method filters target files by file extensions. For instance, files with the extension "xlsx" and files with the extension "pptx" have different data. Meanwhile, files with the extension "xlsx" may have the same data.
- *hash:* This method filters target files by the hash of file data. Generally, the hash is calculated for each file-block. If data is duplicated, then the hash value is the same. The data with the same hash value is compared byte-by-byte, to decide whether the data is same or not.

3 NLPDedup Design

3.1 System Overview

NLPDedup adds the calculation algorithms for file name similarity to the conventional deduplication methods. Figure 1 shows the system overview of NLPDedup. It consists of the modules described below.

(1) Filesystem: NLPDedup processes the files stored in a file system. The file system contains government and company-controlled files, for instance.

(2) Deduplication judgement module: This module judges whether a file can be deduplicated or not. It consists of four parts described as follows.

(a) *Metadata acquiring part*: This part gets the file metadata for deduplication targets from a filesystem.

(b) *Conventional narrowing part*: This part narrows down target files by conventional indicators, such as file extension and file size, etc. When the file size is used as an indicator, NLPDedup sets the ratio of file size difference to from 0.95 to 1.05 according to the previous study [13]. For instance, a 1.00 MB file may be similar to a 1.01 MB file than a 1.10 MB file. Thus, the ratio is important to narrow down the target files.

(c) *File name similarity calculation part*: This part calculates the similarity of file names. To calculate the similarity, NLPDedup employs the Levenshtein distance, character N-gram, and word N-gram. Each algorithm is explained in subsection 3.2.

(d) *Removal judgement processing part*: This part judges the effectiveness of deduplication based on the similarity of file name. The file name similarity threshold is set based on the evaluation results obtained in advance.

(3) Deduplication module: This module deletes duplicated data in accordance with the results of previous part.

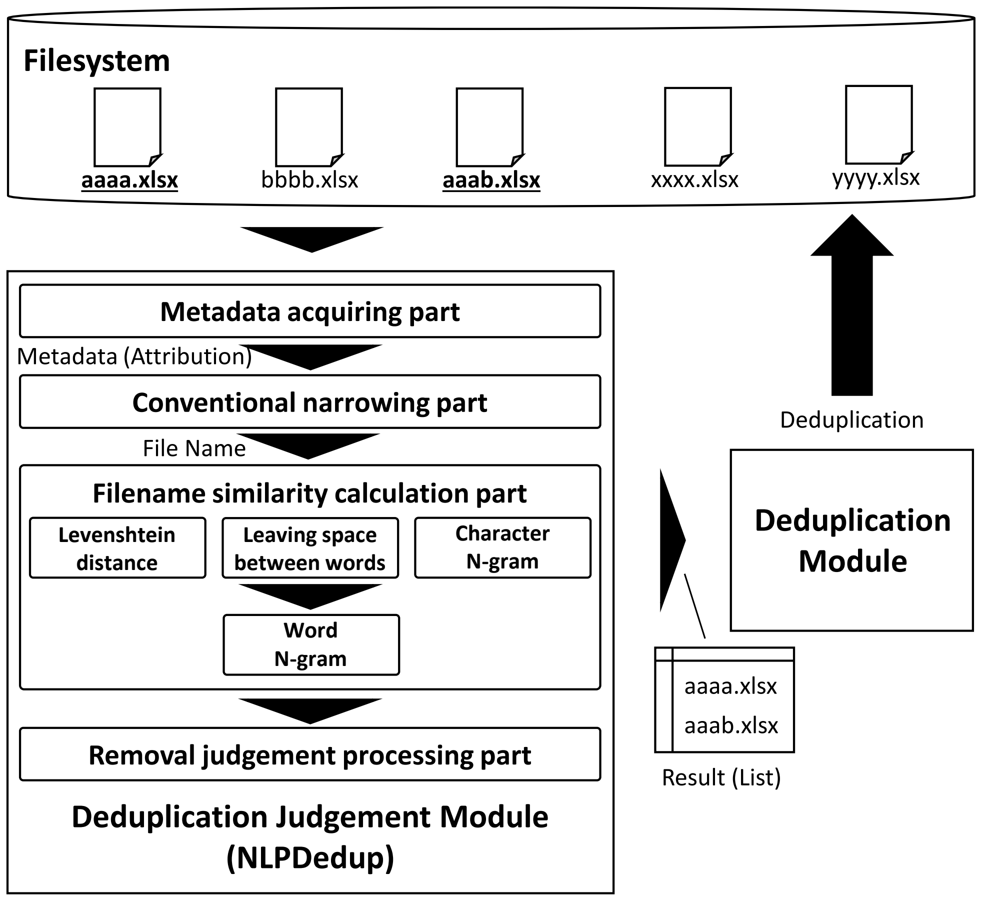

Figure 1: System overview

3.2 Calculation Algorithms of File Name Similarity

To calculate the file name similarity, the algorithms are used described below.

(1) Levenshtein distance: The Levenshtein distance is one of string similarity measures [14]. This indicator is calculated by counting the minimum number of edits (insertions, deletions, or substitutions) required to change one string into the other. To normalize the distance between two file names, the distance is divided by the length of the longer file name. For instance, the distance between a file name "abcd" and a file name "acd" is one, and the calculated distance "1" is divided by four, which is the length of the longer file name "abcd". Since we need the similarity indicator, the normalized value is subtracted from 1.0. Finally, the similarity between the file name "abcd" and the file name "acd" is 0.75. Let the lengths of two file names be m and n, respectively. The time complexity is $O(mn)$.

(2) Character N-gram: The character N-gram is the method of dividing string. In this paper, N-gram is used for calculating string similarity measure. The calculation algorithm divides strings into segmentations of N consecutive characters, then the number of segments of the same contents is calculated. The number is the indicator of character N-gram. The "N" of N-gram means the number of consecutive characters. For example, if the file name "abcd" and the file name "acfe" are processed, then the character uni-gram divides these file names into a, b, c, d and a, c, f, e, respectively. The number of the same characters is counted. In this case, the number is two. To calculate the similarity indicator, this number is divided by the length of the longer file name. Thus, in this case, the similarity indicator is 0.5. Let the lengths of two file names be m and n,

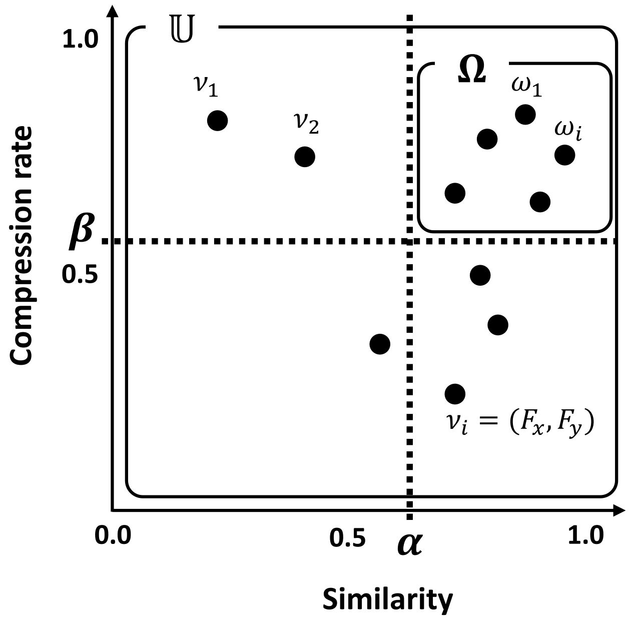

Figure 2: Threshold model of NLPDedup

respectively. The time complexity of computing similarity using character N-gram is $O(m+n)$. **(3) Word N-gram:** The word N-gram is also the method of dividing string. The calculation algorithm of this indicator uses consecutive words instead of characters. The similarity indicator is also normalized by using the same way of the character N-gram.

All the indicator calculation algorithms need word segmentation. However, Japanese is a non-segmented language, and this study focuses on the documents written in Japanese; thus, the file name is also written in Japanese. Therefore, we use "Sudachi" that is a Japanese morphological analyzer [15]. It performs word segmentation with three splitting modes. A mode is the short-unit equivalent of the UniDic dictionary [16]. B mode adds affixes and some compound words to A mode. C mode extracts named entities. In this paper, all modes are applied. Moreover, we use the dictionary SudachiDict [17] for Sudachi. Let the numbers of words in two file names be m and n, respectively. The time complexity of computing similarity using word N-gram is obtained by adding the time complexity of a tokenizer to that of the character N-gram, which is $O(m+n)$.

3.3 Calculating Threshold of File Name Similarity

The indicators for NLPDedup are defined in the previous subsection. In this subsection, threshold calculation algorithms to narrow down target files are described. We developed a model to calculate the threshold for the file name similarity. In this study, the file compression function is used to determine the threshold instead of deduplication. The compression function is widely used and easily set up; thus, the file compression function is better for calculating threshold. Figure 2 shows the threshold calculation model. X-axis is the indicator of file name similarity, and Y-axis is the compression ratio. The file name similarity takes a value from 0.0 to 1.0. The value is calculated by a function described in the previous subsection. The compression ratio takes a value from 0.0 to 1.0. When the value of X-axis is high, the names of files to be processed are similar. When the value of Y-axis is high, two target files can be deduplicated.

The member v_i of the universal set U is two file combinations, such as F_1 and F_2. For instance,

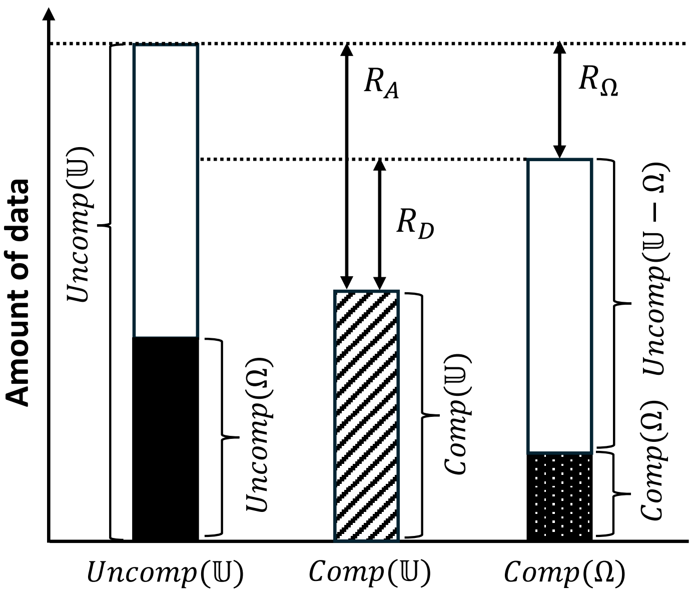

Figure 3: Calculation model for thresholds α

v_1 means a combination of F_1 and F_2, and the file name similarity is 0.2 and the compression ratio is 0.8. Moreover, we define the Ω set as target set. The Ω set contains the member ω_i that satisfies thresholds α for the file name similarity and threshold β for the compression ratio.

This study extracts the member ω_i in Ω set of Figure 2 because the set means high compression ratio achieves and the file names between two files are similar enough. To calculate these thresholds α and β, we focus on the number of ω and the amount of reduction data. These thresholds satisfy the smallest number of ω and the largest amount of reduction data.

First, the universal set U is defined as follows:

$$\mathbb{U} = \{v \mid (F_1, F_2), (F_1, F_3), \dots, (F_i, F_j)\} \tag{1}$$

where F_x means a file, such as file1. That is, the v describes the combination of two files. Next, when all the files in a filesystem are processed, the amount of reduction data R_A is defined as follows:

$$R_A = \sum_{i=1}^{Combin(N,2)} Size(v_i) - \sum_{i=1}^{Combin(N,2)} Size(Comp(v_i)) \tag{2}$$

where the N is the number of files in the filesystem. The $Combin(x, y)$ is a function that calculates the number of combinations, and the $Size(x)$ is also a function that outputs the size of argument. The $Comp(x)$ is a function that compresses a file set v. Next, the members ω of Ω set are defined as follows:

$$\Omega = \begin{Bmatrix} \omega \mid 0.0 < \alpha < Similar(v_i) < 1.0, \\ 0.0 < \beta < Comp(v_i) < 1.0 \end{Bmatrix} \tag{3}$$

where $Similar(x)$ is a function that calculates the file name similarity described in the previous subsection. We define the threshold α and β for the file name similarity and the compression ratio, respectively. Next, when the file sets ω are processed, the amount of reduction data R_Ω is as follows:

$$R_{\Omega} = \sum_{i=1}^{Combin(M,2)} Size(\omega_i) - \sum_{i=1}^{Combin(M,2)} Size(Comp(\omega_i)) \tag{4}$$

where the M is the number of files in Ω set. Next, the differential amount of data reduction between R_A and R_{Ω} is as follows:

$$R_D = R_A - R_{\Omega} \tag{5}$$

Figure 3 shows the model of R_D by using the equation (2), (4), and (5). The X-axis means target methods for comparison, which are all the data without compression, all the data with compression, and compressed data in Ω set. The Y-axis is the amount of data. The left bar is the amount of data in a filesystem without compression, and it includes the amount of Ω set. The middle bar is the amount of all the compressed data in U. The right bar is the amount of the data compressing data in Ω set. R_D is the difference between the middle and right.

To calculate the compression ratio C_R of the threshold model described in Figure 2, the ratio is defined as follows:

$$C_R = 1 - \frac{Size(Comp(\mathcal{V}_i))}{Size(\mathcal{V}_i)} \tag{6}$$

where the C_R is high when the size of compressed data is very small. Thus, to fit the Y-axis of Figure 2, the value is subtracted from 1 as equation (6).

Finally, to determine the threshold α and β, M are calculated using equation (5). In fact, it needs to find the threshold α and β that satisfy the minimum M and the minimum R_D. However, the more M decrease, the more R_D increases. Therefore, we find the cross point of M and R_D by changing the threshold α and β. Next section evaluates how we determine the threshold α and β by using the practical datasets.

4 Evaluation

This section describes evaluation results. For small dataset evaluation, the dataset characteristics and thresholds of the similarity for Kagawa dataset are described in subsections 4.1 and 4.2, respectively. Moreover, for large dataset evaluation, the dataset characteristics and thresholds of similarity for e-gov Data portal dataset are described in subsections 4.3 and 4.4, respectively. These datasets are appropriate for evaluating NLPDedup, because their file names reflect the file contents. The evaluations of both the small and large datasets are conducted using the same method. Finally, we discuss the algorithms and threshold setting in terms of the efficiency and processing performance.

4.1 Kagawa Dataset Characteristics

This subsection describes the small dataset characteristics in terms of the structure of the dataset and similarity, because it needs to show whether the dataset for evaluation is appropriate or not.
(1) Datasets and similarity: To evaluate the proposed method, the datasets are obtained from the Open Data Kagawa web site [18]. The dataset includes 10 directories, 162 'xlsx' files, and the amount of data is 1.88 MB [19]. Each directory contains files of the same category, such as

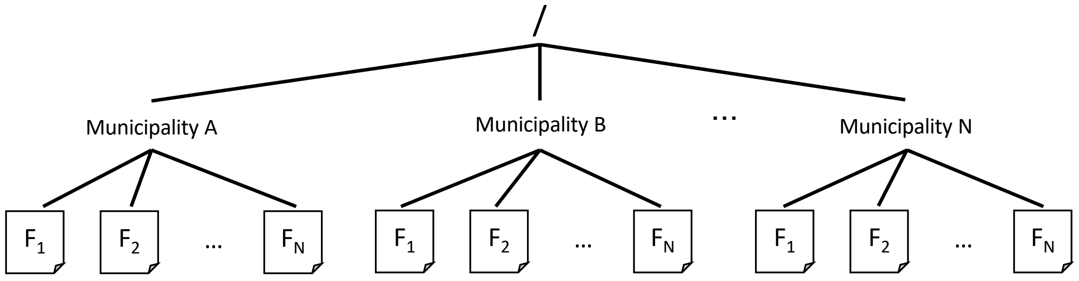

Figure 4: Directory structure

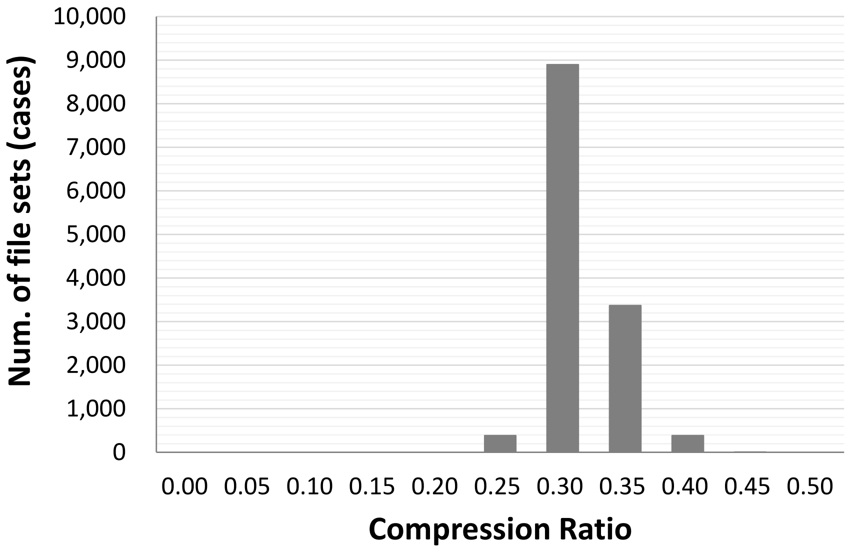

Figure 5: Dataset similarity

the local government bonds. Figure 4 shows an example of directory hierarchy. There are 10 directories under the root directory for each municipality, and the files of municipality data are placed under these directories. The files are 'xlsx' as described above. Moreover, we assume that the conventional narrowing part in Figure 1 runs the process of the file size filtering. The process removes file combinations with the large ratio of file size difference. Based on our study [13], the part filters file combinations by the ratio greater than 0.95 and less than or equal to 1.05. For instance, a file combination of 900 KB file and 1000 KB file is removed because the ratio is 0.90.

To confirm that the dataset can be reduced, the files in the dataset are compressed in pairs. Because a file combination with similar data can achieve a high compressed ratio, the data similarity of two files can be confirmed. Figure 5 shows the similarity of files in the dataset. The horizontal axis indicates compression ratio within a range, e.g. 0.00 means that a compression ratio of a file set is between 0.00 and 0.049. The vertical axis shows the number of file sets. The closer the compression ratio on the horizontal axis is to 0.5, the more data is duplicated. This dataset is distributed with a compression ratio between 0.3 and 0.5. Therefore, the data in the dataset is similar and may have a large amount of duplicated data.

(2) File name and data similarity: To find out the relevance of file name similarity and data similarity, three algorithms described in 3.2 were applied. The Levenshtein distance and the character N-gram are not appropriate for this dataset, because the number of file combinations is many in the range of high file name similarity, and many file combinations with low compression ratio are contained. Therefore, the result of the word tri-gram is shown in this subsection.

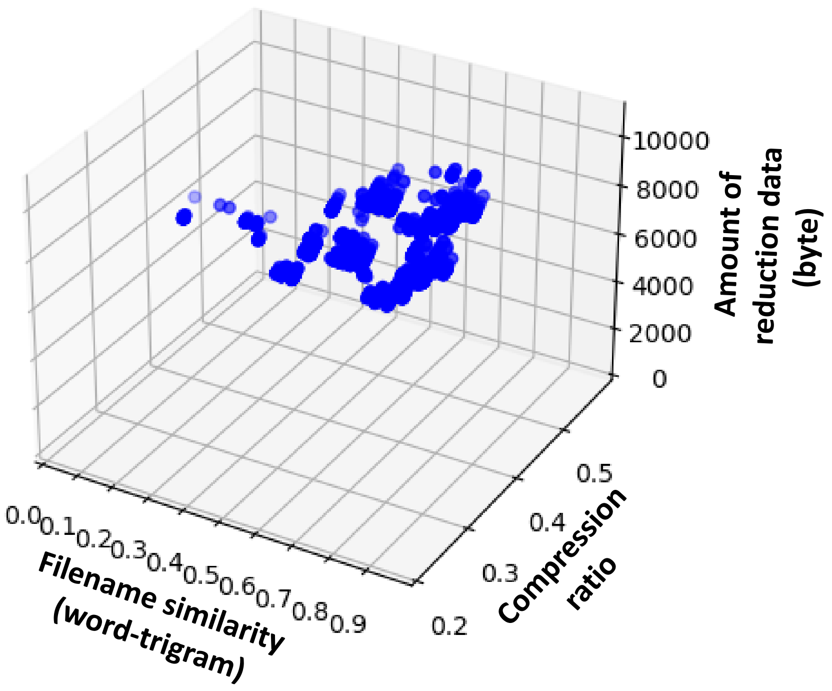

Figure 6: Word tri-gram for C mode segmentation

Figure 6 shows the results of the word tri-gram for C mode segmentation. The X-axis is the file name similarity. The Y-axis is compression ratio. The Z-axis is the amount of reduced data. Many file combinations are distributed within a range of high file name similarity. The results of other algorithms were likewise.

From the results, all the algorithms can narrow down the target file. Particularly, the word N-gram is the best algorithm for extracting file combinations by file name similarity in terms of deduplicated data size. Meanwhile, other algorithms may be applicable if the dataset are different such as large datasets. However, the Levenshtein distance is probably not suitable for large datasets because it has high time complexity shown in subsection 3.2.

As described in subsection 3.3, a threshold of file name similarity is effective for further improvement. The next subsection describes the evaluation results from the standpoint of the threshold.

4.2 Threshold Setting of Kagawa Dataset

This subsection describes the evaluation results of the threshold setting for Kagawa dataset. This subsection focuses on the threshold α described in Figure 3. The threshold β in Figure 3 is fixed to 0.35 according to Figure 5.

(1) Levenshtein distance: Figure 7 shows the number of file combinations and the amount of R_D by using the Levenshtein distance. When the threshold α is set to 0.9, the bar graph and the line graph intersect. Thus, the threshold α is set to between 0.8 and 0.9. Table 1 shows the number of combinations of the Levenshtein distance when threshold α is set to 0.8 and 0.9. To compare the number of file combinations and R_D when threshold α is 0.8 and 0.9, the cases can be reduced to about 1/3; however, the R_D is 2.5x.

(2) Character N-gram: Figure 8 shows the number of file combinations and the amount of R_D by using the character tri-gram. In this case, there are the uni-gram, bi-gram, and tri-gram for the

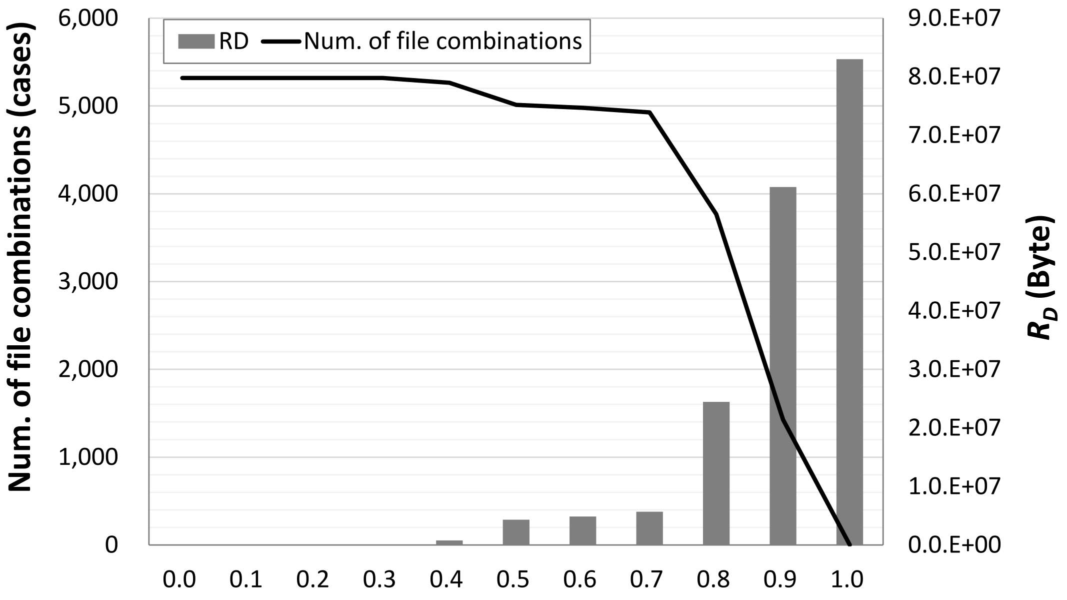

Figure 7: The num. of combinations and R_D for the Levenshutine distance (Kagawa dataset)

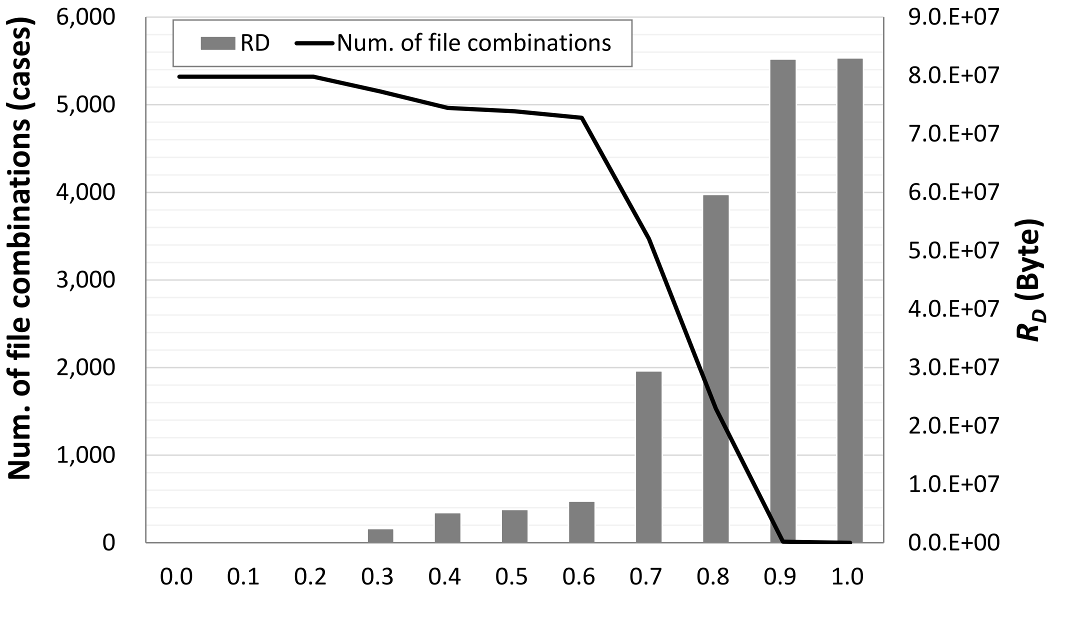

Figure 8: The num. of combinations and R_D for the character tri-gram (Kagawa dataset)

character N-gram. We focus on the tri-gram because the pre-evaluation showed the high efficiency against uni-gram and bi-gram. When threshold α is set to 0.8, the bar graph and the line graph intersect. Table 1 shows the number of combinations of the character tri-gram when threshold α is set to 0.7 or 0.8. To compare the number of file combinations and R_D when threshold α is 0.7 and 0.8, the cases can be reduced to about 1/2; however, the R_D is 2.0x. Therefore, the threshold α between 0.7 and 0.8 is appropriate.

(3) Word N-gram: Figure 9 shows the number of file combinations and the amount of R_D by

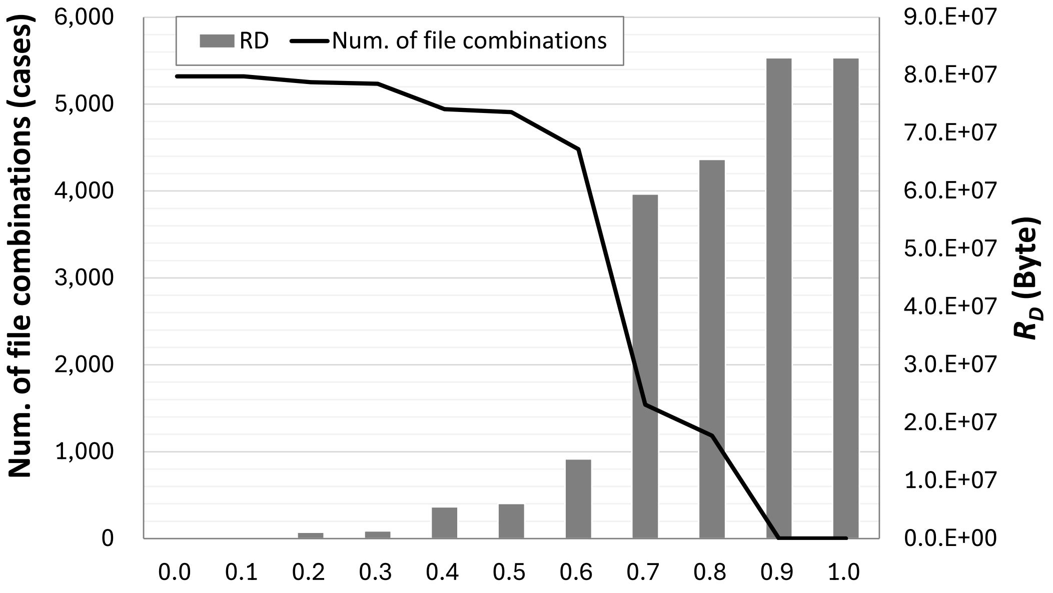

Figure 9: The num. of combinations and R_D for the word tri-gram (Kagawa dataset)

Table 1: The num. of file combinations and R_D for Kagawa dataset

Method	Threshold	Num. of file combinations (case)	R_D (byte)
Levenshtein distance	0.8	3,766	2.4×10^7
	0.9	1,429	6.1×10^7
Character tri-gram	0.7	3,470	2.9×10^7
	0.8	1,533	6.0×10^7
Word tri-gram (C mode)	0.6	4,482	1.4×10^7
	0.7	1,541	5.9×10^7

using the word tri-gram. As described in subsection 3.2, the A, B, and C modes can be applied to the word tri-gram for the segmentation. This study focuses on the C mode that the threshold can be easily set because the distribution of combinations for is sparse compared with other modes. As shown in Table 1, comparing the number of file combinations and R_D when threshold α is 0.6 and 0.7, the cases can be reduced to about 1/3; however, the R_D is 4.0x. Therefore, the threshold α is between 0.6 and 0.7 in this dataset.

4.3 e-gov Data Portal Dataset for Large-scale Evaluation

We found out the effectiveness of small dataset in the previous subsection. Moreover, we assume that NLPDedup is also applied for large dataset because the performance penalty of deduplication may become high. To evaluate the effectiveness of NLPDedup for large datasets, we obtained the

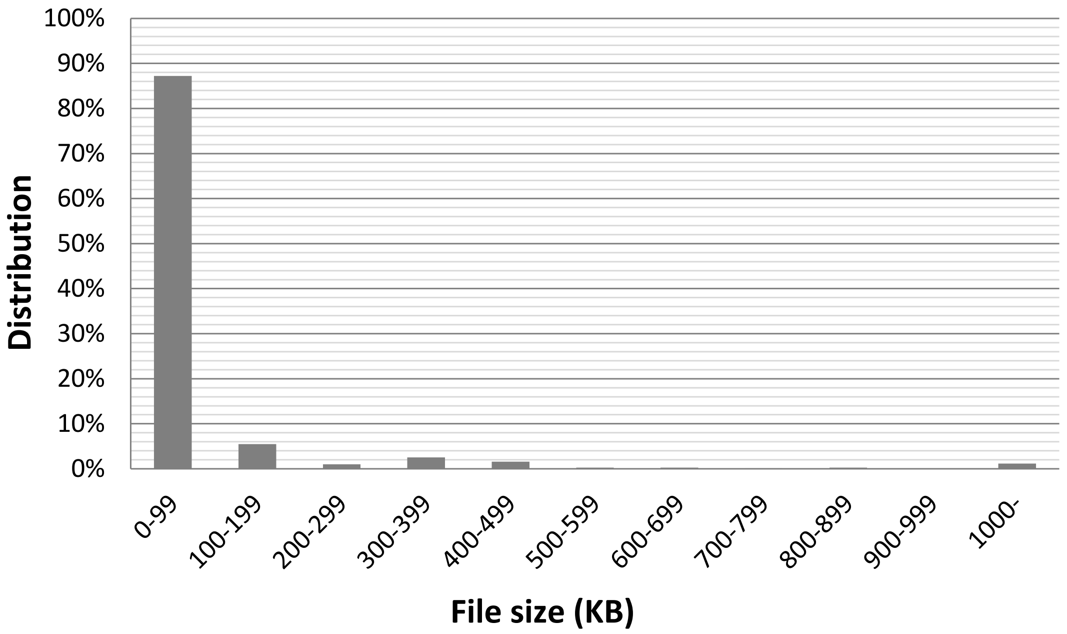

Figure 10: File size distribution of e-gov dataset

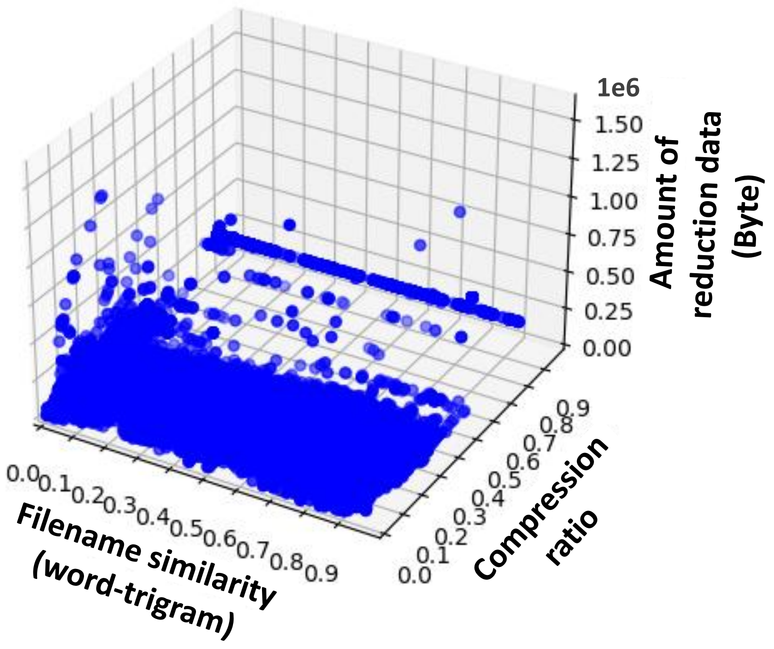

Figure 11: Word tri-gram for C mode segmentation for e-gov dataset

dataset from e-gov Data Portal web site. This subsection describes the dataset characteristics in terms of size and the number of files.

(1) Dataset and similarity: To evaluate NLPDedup for large dataset, the dataset is obtained from the e-gov Data Portal web site [20]. The dataset includes 334 directories, 4748 'xlsx' files, and the total size is 383.76 MB. This dataset has 334 directories under the root directory as shown in Figure 4, and each file in this dataset is placed under these directories. Each directory has files of the same category, such as the budget plans. The maximum file size for the file contained in the dataset is 11,887.4 KB, and the minimum file size is 4.7 KB. This dataset contains large and small

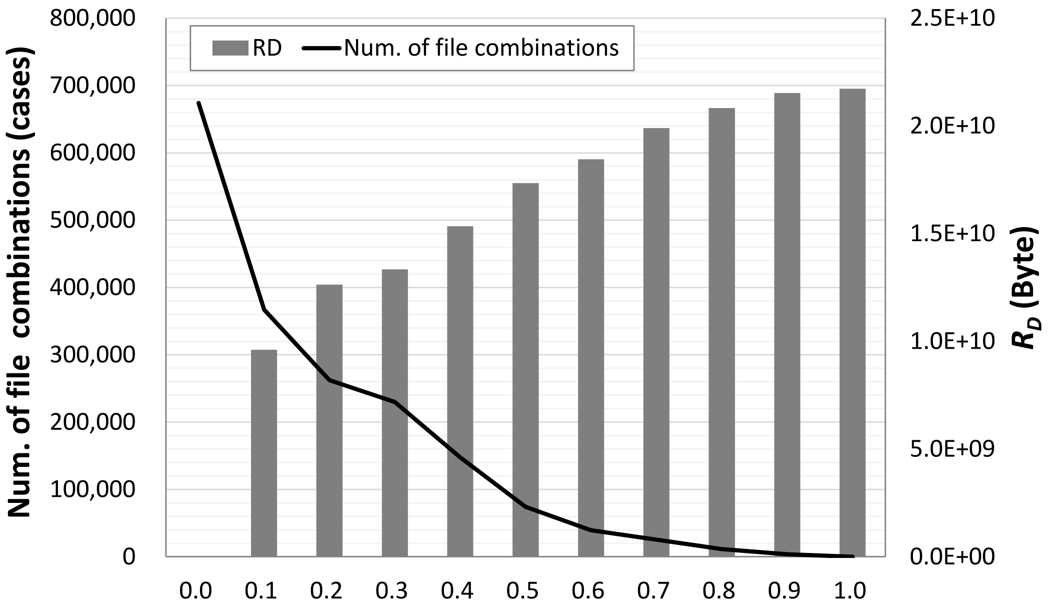

Figure 12: The num. of combinations and R_D for the character tri-gram (e-gov dataset)

files. To find the target files in the dataset, we investigate the file size distribution. Figure 10 shows the file size distribution of the dataset. X-axis indicates file size ranges, and Y-axis shows the ratio of files in a range. According to Figure 10, about 85% of files that are less than 100 KB are stored in the dataset. Thus, the dataset is biased in terms of file size. Furthermore, we assume that the conventional narrowing part filters file combinations as described in subsection 4.1.

(2) File name and data similarity: According to the results from the Kagawa dataset, the Levenshtein distance may not be appropriate for large datasets. Thus, the Levenshtein distance is not evaluated on the dataset. The character N-gram and the word N-gram are evaluated for the file name similarity for the dataset.

The file combination of the character N-gram is distributed across the range of all the file name similarity. Many file combinations are included in the range of low file name similarity. Thus, it is considered that the character N-gram is not appropriate for this dataset.

Figure 11 shows the results of the word tri-gram of C mode segmentation for the dataset. The X-axis indicates the file name similarity. The Y-axis indicates compression ratio. The Z-axis is the amount of reduced data. From the results, many files are distributed across the range of all file name similarity. Therefore, setting to the threshold properly, the number of file combinations and the amount of reduction data can be small instead of fixed threshold. The next subsection describes the evaluation results of e-gov dataset from the standpoint of the threshold.

4.4 Threshold Setting for e-gov Dataset

This subsection describes the evaluation results of the e-gov dataset in terms of the threshold. In this subsection, we apply the same method in subsection 4.2 to evaluate the threshold of e-gov dataset.

(1) Character N-gram: Figure 12 shows the number of file combinations and the amount of R_D by using the character tri-gram. In this case, the uni-gram, bi-gram, and tri-gram for the character N-gram can be applied. However, we focus on the tri-gram because the evaluation results of Kagawa dataset show the high efficiency against the uni-gram and bi-gram. When threshold α

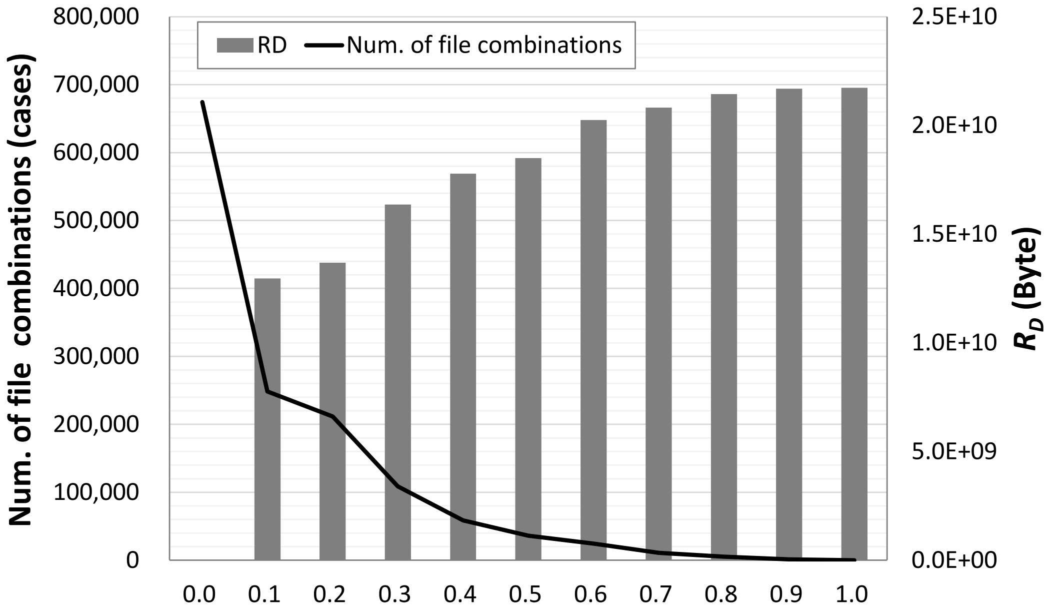

Figure 13: The num. of combinations and R_D for word tri-gram (e-gov dataset)

Table 2: The num. of file combinations and R_D for e-gov dataset

Method	Threshold	Num. of file combinations (case)	R_D (byte)
Character tri-gram	0.1	367,118	0.9×10^{10}
	0.2	262,141	1.3×10^{10}
Word tri-gram (C mode)	0.2	211,548	1.3×10^{10}
	0.3	108,773	1.6×10^{10}

is set to 0.2, the bar graph and the line graph intersect. Table 2 shows the number of combinations for the character tri-gram when threshold α is set to 0.1 and 0.2. To compare the number of file combinations and R_D when threshold α is set to 0.1 and 0.2, these cases can be reduced to about 7/10; however, the R_D is 1.4x. Thus, setting the threshold α to 0.1, the highest efficiency is achieved. Meanwhile, if it needs to reduce the performance penalty, the threshold α is set to higher value, such as 0.5, then the number of file combinations is dramatically reduced.

(2) Word N-gram: Figure 13 shows the number of file combinations and the amount of R_D by applying the word tri-gram. As described in subsection 3.2, the A, B, and C modes can be applied to the word tri-gram for the segmentation. This study focuses on the C mode because it was better according to the evaluation of Kagawa dataset. When threshold α is set to 0.1, the bar graph and the line graph intersect. As shown in Table 2, to compare the number of file combinations and R_D when threshold α is set to 0.2 and 0.3, the cases can be reduced to about 1/2; however, the R_D is 1.2x. Thus, setting the threshold α to 0.2 or 0.3, the highest efficiency is achieved. Meanwhile, if it needs to reduce the performance penalty, the threshold α should be set to higher value.

4.5 Discussion

From the evaluation results of subsections 4.1 and 4.2, NLPDedup can narrow down target files by using file name similarity. Thus, NLPDedup can reduce effectively the amount of data when the appropriate threshold is applied.

If high performance for deduplication process is needed, then the threshold of file name similarity for NLPDedup is set to high because the files to be processed are narrowed down. Meanwhile, if the high deduplication ratio is needed, then the low threshold value is set, such as setting the threshold just before the intersection of the bar graph and the line graph. In this way, NLPDedup can vary the threshold; thus, the policy of threshold can be set depending on the use cases.

According to subsections 4.1 and 4.2, NLPDedup is effective for small datasets. Moreover, NLPDedup is also effective for large datasets according to subsections 4.3 and 4.4. In the case of small datasets, the threshold of file name similarity can be set to a fixed value. Meanwhile, in the case of large datasets, the threshold can be set to variable values. For instance, if high performance is required for NLPDedup processing, the threshold is set to a high value. From the results, NLPDedup can narrow down target files for small and large datasets. NLPDedup can reduce the number of file combinations to be processed, and the performance penalty can be mitigated. Examples of effective datasets for NLPDedup are government data, enterprise data and academic archives. Meanwhile, datasets generated by end users are not suitable, because their file names are unstructured.

We evaluated three algorithms for NLPDedup, which are the Levenshtein distance, character N-gram, and word N-gram. Consequently, the word N-gram is the best algorithm in these algorithms because the small and large datasets can be narrowed down efficiently. In addition, its time complexity is lower than that of the Levenshtein distance. Even though word segmentation incurs additional computation cost, word N-gram is more effective than character N-gram in terms of filtering. According to Table 2, the word tri-gram can reduce the number of file combinations while maintaining the amount of deduplicated data. In conclusion, NLPDedup is effective not only for small datasets but also for large datasets. It is important for deduplication of small and large datasets by NLPDedup to set the appropriate threshold of file name similarity.

5 Related Work

InftyDedup focuses on cloud backup deduplication [21]. By sending deduplicated data to cloud storage, it can reduce backup costs by 26%–44%. Single-instance storage is one of the deduplication methods for files stored in a file storage server. The single-instance storage processes for each file. Meanwhile, fixed-length block deduplication and variable-length block deduplication process sub-file; thus, it is not necessary that the data in files to be deduplicated is completely same. Generally, the deduplication rate of block deduplication is higher than single-instance storage. However, the deduplication rate of fixed-block and variable-block is 70 % and 78 % against single-instance storage, respectively [11]. The paper shows that the deduplication process of block-wise segmentation is effective, but it is also expensive. DedupSearch applies keyword search to the deduplication process [22]. DedupSearch accelerates the deduplication process by searching one or more byte strings in block deduplication. DedupSearch can determine string matches for each data chunk by reading the chunk once. GogetaFS also focuses on I/O load [23]. By introducing a mapping table for deduplicated data, it can reduce the number of I/O issued by 38%.

Previous deduplication studies for backup systems do not focus on the file name for narrowing down target files. Meanwhile, NLPDedup adds the NLP process to the conventional methods; thus, NLPDedup boosts the performance of methods such as DedupSearch and GogetaFS, because it can reduce the number of I/Os issued during deduplication processes.

6 Conclusion

This paper described that file name similarity is effective to narrow down target files for reducing the number of read I/O issued. Our proposed method, called NLPDedup, applies the NLP process to the conventional deduplication methods. NLPDedup processes a file name to find the data similarity, and it applies the Levenshtein distance, character N-gram, and word N-gram as indicators of file name similarity. This paper provided an overview of NLPDedup and a formula of the threshold determination for file name similarity. We evaluated the NLPDedup by practical datasets to confirm the efficiency and the formula. In this study, these datasets are small and large for evaluation. From evaluation results, NLPDedup can narrow down target files, and the threshold can be determined by using the proposed formula. NLPDedup can improve deduplication efficiency even for large datasets. Moreover, the threshold depends on the datasets of use cases for each algorithm of NLP. In the future, we will implement the system in an actual deduplication process to verify its effectiveness, and we will search for more effective ways to refine our search results.

7 References

[1] P. Shilane et al., “Delta Compressed and Deduplicated Storage Using Stream-Informed Locality,” Proc. 4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '12), 2012, pp. 1-5.

[2] Y. Zhang et al., “LoopDelta: Embedding Locality-aware Opportunistic Delta Compression in Inline Deduplication for Highly Efficient Data Reduction,” Proc. 2023 USENIX Annual Technical Conference (USENIX ATC '23), 2023, pp. 133-148.

[3] L. L. You, K. T. Pollack, and D. D. E Long, “Deep Store: an archival storage system architecture,” Proc. 21st International Conference on Data Engineering (ICDE'05), IEEE, 2005, doi: 10.1109/ICDE.2005.47

[4] P. Kulkarni et al., “Redundancy Elimination Within Large Collections of Files,” Proc. 2004 USENIX Annual Technical Conference (USENIX ATC '04), 2004, pp. 59-72.

[5] P. Shilane et al., “WAN Optimized Replication of Backup Datasets Using Stream-Informed Delta Compression,” Proc. 10th USENIX Conference on File and Storage Technologies (FAST '12), 2012, pp. 49-63.

[6] N. Jain et al., “TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization,” Proc. 4th USENIX Conference on File and Storage Technologies (FAST '05), 2005, pp. 281-294.

[7] M. Dutch, “Understanding data deduplication ratios,” SNIA, Jun. 2008,

https://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf

[8] J. Qiu et al., “Light-Dedup: A Light-weight Inline Deduplication Framework for Non-Volatile Memory File Systems,” Proc. 2023 USENIX Annual Technical Conference (USENIX ATC '23), 2023, pp. 101-116.

[9] I. Kotlarska et al., “InftyDedup: Scalable and Cost-Effective Cloud Tiering with Deduplication,” Proc. 21st USENIX Conference on File and Storage Technologies (FAST '23), 2023, pp. 33-48.

[10] X. Zou et al., “Building a High-performance Fine-grained Deduplication Framework for Backup Storage with High Deduplication Ratio,” Proc. 2022 USENIX Annual Technical Conference (USENIX ATC '22), 2022, pp. 19-36.

[11] D. T. Meyer et al., “A Study of Practical Deduplication,” Proc. 9th USENIX Conference on File and Storage Technologies (FAST '11), 2011, pp. 1-13.

[12] H. Yokoyama et al., “NLPDedup: Using Natural Language Processing for Data Deduplication,” Proc. IIAI AAI 2024 16th International Congress on Advanced Applied Informatics (IIAI-AAI '24), 2024, p. 115-120, 10.1109/IIAI-AAI63651.2024.00031.

[13] H. Yokoyama et al., “A Method of Duplicate Data Detection by File Names as Feature Value,” The 86th National Convention of IPSJ, 2024, pp.459-460.

[14] J. Lu et al., “String similarity measures and joins with synonyms,” Proc. 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13), ACM, 2013, doi: 10.1145/2463676.2465313.

[15] WAP Tokushima NLP Resources, “Sudachi,” https://worksapplications.github.io/Sudachi/

[16] National Institute for Japanese Language and Linguistics, “Electronic Dictionary with Uniformity and Identity. UniDic,” https://clrd.ninjal.ac.jp/unidic/

[17] WAP Tokushima NLP Resources, “SudachiDict,” https://github.com/WorksApplications/SudachiDict/.

[18] Kagawa prefecture, “Open Data KAGAWA,” https://opendata.pref.kagawa.lg.jp/dataset/.

[19] H. Yokoyama et al., “Acceleration of Deduplication Processing Using File Name Similarity,” 2023 Shikoku-Section Joint Convention Record of the Institutes of Electrical and Related Engineers, 2023, p.199.

[20] Digital Agency, “e-gov Data Portal,” https://data.e-gov.go.jp/info/ja.

[21] I. Kotlarska et al., “InftyDedup: Scalable and Cost-Effective Cloud Tiering with Deduplication,” Proc. 21th USENIX Conference on File and Storage Technologies (FAST '23), 2023, pp. 33-48.

[22] N. Elias et al., “DedupSearch: Two-Phase Deduplication Aware Keyword Search,” Proc.

20th USENIX Conference on File and Storage Technologies (FAST '22), 2022, pp. 233-246.

[23] Y. Pan et al., “Don't Maintain Twice, It's Alright: Merged Metadata Management in Deduplication File System with GogetaFS,” Proc. 23th USENIX Conference on File and Storage Technologies (FAST '25), 2025, pp. 479-495.