Cleansing Data for Mining and Warehousing

Mong Li Lee Hongjun Lu Tok Wang Ling Yee Teng Ko

School of Computing
National University of Singapore
{leeml, luhj, lingtw}@comp.nus.edu.sg

Abstract. Given the rapid growth of data, it is important to extract, mine and discover useful information from databases and data warehouses. The process of data cleansing is crucial because of the "garbage in, garbage out" principle. "Dirty" data files are prevalent because of incorrect or missing data values, inconsistent value naming conventions, and incomplete information. Hence, we may have multiple records refering to the same real world entity. In this paper, we examine the problem of detecting and removing duplicating records. We present several efficient techniques to pre-process the records before sorting them so that potentially matching records will be brought to a close neighbourhood. Based on these techniques, we implement a data cleansing system which can detect and remove more duplicate records than existing methods.

1 Introduction

Organizations today are confronted with the challenge of handling an ever-increasing amount of data. In order to respond quickly to changes and make logical decisions, the management needs rapid access to information in order to research the past and identify relevant trends. These information is usually kept in very large operational databases and the easiest way to gain access to this data and facilitate strategic decision making is to set up a data warehouse. Data mining techniques can then be used to find "optimal" clusterings, or interesting irregularities in the data warehouse because these techniques are able to zoom in on interesting sub-parts of the warehouse.

Prior to the process of mining information in a data warehouse, **data cleansing** or **data scrubbing** is crucial because of the "garbage in, garbage out" principle. One important task in data cleansing is to **de-duplicate** records. In a normal client database, some clients may be represented by several records for various reasons: (1) incorrect or missing data values because of data entry errors, (2) inconsistent value naming conventions because of different entry formats and use of abbreviations such as 'ONE' vs '1', (3) incomplete information because data is not captured or available, (4) clients do not notify change of address, and (5) clients mis-spell their names or give false address (incorrect information about themselves). As a result, we encounter situations where several records may refer to the same real world entity while not being syntactically equivalent. We can treat a set of records that refer to the same entity in two ways. We can view one of the records as correct and the rest of the records as duplicates containing erroneous information. Then the objective is to cleanse the database of the duplicate records [6, 2]. Alternatively, we can view each matching record as a partial source of information. Then the objective is to merge the duplicate records to obtain one record with more complete information.

In this paper we hold the latter view when we examine the problem of detecting and removing duplicating records. We present several novel techniques to pre-process the records before sorting them so that potentially matching records will be brought to a close neighbourhood subsequently. This will enable more matching records to be detected and removed. The pre-processing techniques include scrubbing data fields using external source files to remove typographical errors and the use of abbreviations, tokenizing data fields and then sorting the tokens in the data fields to solve the different field entry format problem which always exists in dirty data files but has been neglected by existing methods. We also introduce the use of field weightage to compute similarity among records. Accuracy is further improved with the help of external source files. Based on these techniques, we implement a data cleansing system which is able to detect and remove duplicate records than existing methods.

The rest of the paper is organized as follows. Section 2 gives a motivating example and surveys related works. Section 3 describes our proposed data cleansing methodology. Section 4 discusses the implementation and time complexity of our system, and finally we conclude in Section 5.

2 Motivation

To remove duplicated records from a dataset, the main consideration is how to decide that two records are duplicate? We need to compare records to determine their degree of similarity, which implies that corresponding fields in the records has to be compared. The comparison of fields to determine whether or not two syntactic values are alternative representations of the same semantic entity is also known as the **field matching problem** [5].

Record	EmpNo	Name	Address
1	142625M	Liu Hang Xiang	1020 Jalan Bandar Lamma, Industrial Park 3, West Malaysia
2	142725M	Mr. Liu H.X.	Ind Park 3, 1020 Jalan Bandar Lama, Malaysia

Table 1. Example of two duplicate records.

Table 1 shows two records, Record 1 and Record 2. At first glance, all the field values in both records look different. On closer examination, we note that the EmpNo in Record 1 and Record 2 are very similar except for a digit difference. We observe that "Liu" is common in the Name field of Record 1 and Record 2 and "H.X." in Record 2 seems to be an abbreviation of "Hang Xiang" in Record 1. If the address of Record 2 is reorganized as {*1020 Jalan Bandar Lamma,* *Ind Park 3, Malaysia*}, we find that the Address of Record 1 and Record 2 are actually the same except for a typographical error {*Lamma*} in Record 1 and a missing word {*West*} in Record 2. Moreover, abbreviation {*Ind*} has been used in Record 2 instead of {*Industrial*}. Since the EmpNo, Name and Address field values of Record 1 and 2 are very similar to each other, we may conclude that Record 1 and Record 2 are most likely to be duplicates and they refer to the same employee in the real world. The differences in the Name and Address field values in Record 1 and 2 are typical of **different field entry format** problem.

There has been little research on the field matching problem although it has been recognized as important in the industry. Published work deals with domain-specific cases such as the Smith-Waterman algorithm for comparing DNA and protein sequences [7], and variant entries in a lexicon [4]. [2] use domain specific equational axioms to determine if two tuples are equivalent. [5] gives a basic field matching algorithm based on matching strings and a recursive algorithm to handle abbreviations. However, the former algorithm does not handle abbreviation while the latter has quadratic time complexity.

The standard method of detecting **exact** duplicates in a database is to sort the database and check if neighbouring records are identical [1]. The most reliable way to detect approximate duplicates is to compare every record with every other record in the database. But this is a very slow process which requires N(N-1)/2 record comparisions, where N is the number of records in the database. [2] proposed a **Sorted Neigbourhood Method (SNM)** to detect approximate duplicates by first sorting the database on a chosen application-specific key such as {Name, Address} to bring "potentially matching" records to within a close neighbourhood. This key is a sequence of a subset of attributes, or substrings within the attributes, which has sufficient discriminating power in identifying likely candidates for matching. There is no rule specifying how the key should be designed. We can design a key which concatenates the first 3 digits in EmpNo and the first 5 consonants in Name. Next, pairwise comparisons of nearby records are made by sliding a window of fixed size over the sorted database. Suppose the size of the window is w records, then every new record entering the window is compared with the previous w-1 records to find "matching records". The first record in the window slides out of the window.

SNM is obviously faster since it requires only wN comparisons. However, the effectiveness of this approach depends on the quality of the chosen keys which may fail to bring possible duplicate records near to each other for subsequent comparison. For example, if we choose the Address field in Table 1 to be the key to sort the database, then Record 1 and Record 2 will be very far apart after sorting because the address field value of Record 1 starts with "1020" while that of Record 2 starts with "Ind". If we choose the Name field to be the sort key, then Record 1 and Record 2 will be very close after sorting since both their name field values start with "Liu".

The **Duplication Elimination SNM (DE-SNM)** [3] improves the results of SNM by first sorting the records on a chosen key and then dividing the sorted records into two lists: a duplicate list and a no-duplicate list. The duplicate list contains all records with exact duplicate keys. All the other records are put into the no-duplicate list. A small window scan is performed on the duplicate list to find the lists of matched and unmatched records. The list of unmatched records is merged with the original no-duplicate list and a second window scan is performed. But the drawback of SNM still persists in DE-SNM.

In general, the duplicates elimination problem is difficult to handle both in scale and accuracy. Our proposed approach aims to increase the accuracy by first pre-processing the records so that subsequent sorting will bring potentially matching records to a close neighbourhood. In this way, the window size can be reduced which improves processing time. Finally, we note that while there are a few data cleansing software in the industry, most companies do not disclose the details of how it's done.

3 Proposed Cleansing Methodology

Our approach to cleansing a database comprises of several steps.

1. **Scrub dirty data fields.** This step attempts to remove typographical errors and abbreviations in data fields. This will increase the probability that potentially matching records be brought closer after sorting which uses keys extracted directly from the data fields.
2. **Sort tokens in data fields.** Characters in a string can be grouped into meaningful pieces. String values in data fields such as Name and Address can be split into meaningful groups, called **tokens**, which are then sorted.
3. **Sort records.**
4. **Comparison of records.** A window of fixed size is moved through the sorted records to limit the comparisons for matching records. Field weightage is used to compute the degree of similarity between two records.
5. **Merge matching records.** Matching record are treated as a partial source of information and merged to obtain a record with more complete information.

Steps 1 and 2 are not found in existing cleansing methods. These additional steps enhance the possibility that matching records will be brought closer during the sorting. The following subsections elaborates on steps 1, 2 and 4.

3.1 Scrubbing Dirty Data Fields

Existing data cleansing techniques such as the SNM and the DE-SNM are highly dependent on the key chosen to sort the database. Since the data is dirty and the keys are extracted directly from the data, then the keys for sorting will also be dirty. Therefore, the process of sorting the records to bring matching records together will not as effective. A substantial number of matching records may not be detected in the subsequent window scan.

Data in records are "dirtied" in various ways. It is common to find data entry errors or typing mistakes in name and address fields. Such **typographical**

errors causes the data to be incorrect or contain missing values. These fields may have different **entry format** as illustrated in Table 1. **Abbreviations** are often used to speed up data entry. The effectiveness of any de-duplicating method is to first remove such dirty data in the record fields.

Suppose we have a record with entry *ACER TECHNOOLGY PTE LTD* in its Company Name Field. There may be some typographical error in this field which cannot be corrected by a spelling checker because special names such as the name of a person or a company cannot be found in any dictionaries. For example, *ACER* is not spelled wrongly because it is a company name but *TECHNOOLGY* has a typographical error. Abbreviations such as *TECH.* for *TECHNOLOGY* may also be used. To ensure the correctness of data in the database, we use **external source files** to validate the data and resolve any data conflicts. The external source files contain information in record format. each record will have fields as shown in Table 2. Such external source files can be obtained from National Registries such as the Registry of Birth, Registry of Companies etc, which would contain more accurate and complete information on a person or company.

In Table 2, a particular person's information is contained in only one record. This external source file can be used to format and correct the information in a "dirty" database. We note that there exists a functional dependency $SSNO \rightarrow Name, Age, Sex$ in our example external source file. $SSNO$ is unique and is called the **key** field. This feature in the external source file may be used to enforce any functional dependencies between the fields in the database. Fields in the source files should correspond to fields in the database and this correspondence have to be provided by users. Formatting of the fields in the "dirty" database will be carried out according to key field in the external source file. Table 3 shows an example "dirty" record in the database. During the scrubbing process, the system will find the SSNO of this record in the external source file (Table 2). It will then change the values of the Name and Age fields of of the "dirty" record (Table 3) to the corresponding field values of the equivalent record in the external source file (Table 2). Table 4 shows the cleansed record with the Name field value re-formatted and the Age value corrected. With this step, we can guarantee the correctness of data as well as standardize the entry format in the database.

There are two possible scenarios for errors in the SSNO of the dirty database:

1. The wrong SSNO does not exist in external source file.
   In this case, the system would inform the user of the error.
2. The SSNO is the SSNO of another person.
   Here, the system should calculate the similarity between the record in the database and those in the external source file. We develop a method to compute the similarity between two records by using field weightage. This method (details in section 3.3) can be used to calculate the similarity between a record in the database and a matching record in the external file. The field values in the database record will only be re-formatted or corrected if

the computed similarity exceed certain value. Otherwise, the system would prompt the user whether or not to format the record in the database.

SSNO	Name	Age	Sex
0273632T	Koh Yiak Heng	43	M
3635290Y	Tan Kah Seng	16	M
5927356K	Vivian Chua	25	F

Table 2. Example of an external source file.

SSNO	Name	Age	Sex
0273632T	Koh Y.H.	42	M

Table 3. "Dirty" record in the database.

SSNO	Name	Age	Sex
0273632T	Koh Yiak Heng	43	M

Table 4. "Cleaned" record in the database.

3.2 Tokenizing and Sorting Data Fields

We have seen how a key chosen for sorting the database records plays an important role in bringing potentially matching records to within a window. This key can also cause the matching records to become further apart and hence reduce the effectiveness of the subsequent comparison phase. Table 5 shows three records in a database. If we choose the Address field in Table 5 to be the key to sort the database, then Record 1 and Record 2 will be very far apart after sorting because the address field value of Record 1 starts with a numeric string "1020" while that of Record 2 starts with "Industrial".

We observe that characters in a string can be grouped into meaningful pieces. We can often identify important components or **tokens** within a Name or Address field by using a set of delimiters such as space and punctuations. Hence, we can first tokenize these fields and then sort the tokens within these fields. For example, we obtain the tokens {Liu Kok Hong} in the Name field of Record 1 in Table 5. After sorting these tokens, we will obtain {Hong Kok Liu}. Table 6 shows the resulting database.

Records will now be sorted based on the sorted tokens in the selected key field. If the user chooses to use the Address field to sort the database, then the order of the records in the database will be 3, 2, 1. However, if the user selects the Name field to sort the database, then the order of the records in the database will be 2, 1, 3. Users can also choose to use {Name, Address} to sort the database. In this case, the system will make two pass on the database. It will first sort the records according to the Name field and remove any duplicate records. Then it will sort the database according to the Address field and remove any duplicate records. Information in the duplicate records are merged to obtain a record with more complete information. Note that if a field contains digits and character strings, then we need to separate the character string tokens and digit tokens. Otherwise, a record containing an address with a house number will never be close to another record with the same address but without the house number. Furthermore, users should choose fields which contains representative information of the record. For example, using the Sex field to sort the database will not be able to bring matching records close to each other since there are a lot of records containing same value in this field.

Record	Name	Address	Sex
1	Liu Kok Hong	1020 Jalan Bandar Lama, Industrial Park 3, Malaysia	M
2	Liu K.H.	Industrial Park 3, 1020 Jalan Bandar Lama, Selangor Darul Ehsan, Malaysia	M
3	Yap Kooi Shan	Blk 33 Marsiling Ind. Estate, #07-03, Singapore 130037	F

Table 5. Unsorted database

Record	Name	Address	Sex
1	Hong Kok Liu	3 1020 Bandar Industrial Jalan Lama Malaysia Park	M
2	H K Liu	3 1020 Bandar Darul Ehsan Ind. Jalan Lama Selangor	M
3	Kooi Shan Yap	03 07 33 130037 Blk Estate Ind. Marsiling Singapore	F

Table 6. Database with fields tokenised and sorted

3.3 Comparing Records

After the records in the database has been sorted, a window of fixed size w is moved through the records to limit comparisons of potentially matching records to those records in the window. Every new record entering the window is compared with the previous $w-1$ records to find matching records. The first record in the window slides out of the window.

An efficient method is required to compare two records to determine their degree of similarity. We introduce the concept of **field weightage** which indicates the relative importance of a field to compute the **degree of similarity** between two records. The Name field obviously have a higher weightage than Sex field since because name is more representative of a record than sex. The field weightage is provided by users and the sum of all field weightages should be equal to 1. For example, if the user want to eliminate duplicate records based on the Name and Address fields equally, then they should assign a weightage of 0.5 to each of these two fields and 0 for the other fields in the record. Thus, records with same Name field and Address field will be considered as duplicates.

The process of computing the similarity between two records starts with comparing the sorted tokens of the corresponding fields. The tokens are compared using exact string matching, single-error matching, abbreviation matching and prefix matching. Based on the field token comparison results, the similarity between the entire field is computed. Finally, the record similarity can be computed from the fields similarity and the fields weightage. This is given in the following two propositions.

Proposition: Field Similarity

Suppose a field in Record X has tokens $t_{x_1}, t_{x_2}, ..., t_{x_n}$ and a corresponding field in Record Y has tokens $t_{y_1}, t_{y_2}, ..., t_{y_m}$. Each token $t_{x_i}, 1 \leq i \leq n$ is compared with tokens $t_{y_j}, 1 \leq j \leq m$. Let $DoS_{x_1}, ..., DoS_{x_n}, DoS_{y_1}, ..., DoS_{y_m}$ be the maximum of the degree of similarities computed for tokens $t_{x_1}, ..., t_{x_n}$, $t_{y_1}, ..., t_{y_m}$ respectively. Then field similarity for Record X and Y $Sim_F(X,Y)$ is given by $(\sum_{i=1}^{n} t_{x_i} + \sum_{i=1}^{m} t_{y_i})/(n+m)$.

Proposition: Record Similarity

Suppose a database has fields $F_1, F_2, ..., F_n$ with field weightages $W_1, W_2, ..., W_n$ respectively. Given records X and Y, let $Sim_{F_1}(X,Y), ..., Sim_{F_n}(X,Y)$ be the field similarities computed. Then record similarity for X and Y is given by the expression $\sum_{i=1}^{n} Sim_{F_i}(X,Y) * W_i$

We can have a rule that two records with record similarity exceeding a certain threshold such as 0.8 are duplicates and therefore, should be merged. While it is straightforward to check whether two tokens are exactly the same, it is not sufficient because of the existence of typographical errors, use of abbreviations etc. We need to consider single-error matching, abbreviation matching and substring matching when comparing tokens to calculate the degree of similarity. If two tokens are an exact match, then they have a degree of similarity of 1. Otherwise, if there is a total of x characters in the token, then we deduct $\frac{1}{x}$ from the maximum degree of similarity of 1 for each character that is not found in the other token. For example, if we are comparing tokens "cat" and "late", then $DoS_{cat} = 1 - \frac{1}{3} = 0.67$ since the character c in "cat" is not found in "late" and $DoS_{late} = 1 - \frac{2}{3} = 0.33$ since the characters l and e are not found in "cat". We shall now elaborate on the various matching techniques and how the degree of similarity of tokens are obtained.

1. **Exact string matching**
   The standard *strcmp()* function will return 1 if two tokens are exactly the same, else return 0.
2. **Single-error matching**
   Single-error checking includes checking for additional characters, missing characters, substituted characters and transposition of adjacent characters. Table 7 shows resulting degree of similarities when we compare the tokens "COMPUPTER", "COMPTER", "COMPUTOR", "COMPUTRE" to the token "COMPUTER".
3. **Abbreviation matching**
   An external source file containing the abbreviations of words is needed. Table 8 shows an example abbreviation file. A token A is a possible abbreviation of token B only if all the characters in A are contained in B and these char-

acters in A appear in the same order as in B. If a token is found to be an abbreviation of another, then they have a similarity of degree 1.

4. **Prefix substring matching**
Here, we look for two similar tokens where one is a leading substring of the other. For example, "Tech." and "Technology", or "Int." and "International". Note that $DoS_{Tech} = 1$ since all the characters in "Tech." are found in "Technology" while $DoS_{Technology} = 0.4$ since there are 6 characters in "Technology" that are not found in "Tech". If a substring does not occur at the beginning of a token, then the two token may not be too similar. For example, "national" and "international" and we assign a similarity of degree of 0.0 for both these tokens.

Token 1	Token 2	DoS_{Token1}	DoS_{Token2}
COMPUTER	COMPUPTER	1.0	0.89
COMPUTER	COMPTER	0.88	1.0
COMPUTER	COMPUTOR	0.88	0.88
COMPUTER	COMPUTRE	1.0	1.0

Table 7. Single-error matching

Abbreviation	Word
SVCS	Services
PTE	Private
LTD	Limited

Table 8. Example of an abbreviation file

4 Data Cleansing System - Implementation and Performance

We implemented a data cleansing system in C on the UNIX and tested our system with an actual company dataset of 856 records. Each record has seven fields: Company Code, Company Name, First Address, Second Address, Currency Used, Telephone Number and Fax Number. Manual inspection of the dataset reveals 40 duplicate records. Typical problems in this dataset include records with empty Company Code or Address, matching records with different Company Code, typographical errors and abbreviations. The fields which contains representative information of a record and are most likely able to distinguish the records are Company Name, First Address and Second Address. We merged the First Address and Second Address fields because almost half the number of records have empty First Address.

It is possible that duplicate records are not detected and similar records which do not represent the same real world entity are treated as duplicates. These incorrectly paired records are known as **false-positives**. We obtain the following results when we run our system on the company dataset with a window size of 10:

1. **Misses.** The system failed to detect 5 individual records. That is, it has 12.5 % misses or 87.5 % true-positives.
2. **False-positives.** The system incorrectly matched 1 record. That is, it has 0.12 % false-positives.

The results show that our system is able to detect and remove the majority of the duplicate records with minimal false-positives. The additional pre-processing steps of scrubbing the data fields using external source files, tokenizing and sorting the data fields enables the subsequent sorting step to bring more potentially matching records to a close neighbourhood. An mathematical analysis of our system's time complexity shows that although these pre-processing steps may take extra time, they are not exponential.

5 Conclusion

We have examined the problem of detecting and removing duplicating records. We presented several efficient techniques to pre-process the records before sorting them so that potentially matching records will be brought to a close neighbourhood subsequently. These techniques include scrubbing data fields using external source files to remove typographical errors and the use of abbreviations, tokenizing data fields and then sorting the tokens in the data fields. These pre-processing steps, which have been neglected by existing methods, are necessary if we want to detect and remove more duplicate records. We also proposed a method to determine the degree of similarity between two records by using field weightage. We implemented a data cleansing system and the preliminary results obtained has been encouraging. Ongoing work involves testing the system's scalability and accuracy with real-world large data set.

References

1. D. Bitton and D.J. DeWitt. Duplicate record elimination in large data files. *ACM Transactions on Database Systems*, 1995.
2. M. Hernandez and S. Stolfo. The merge/purge problem for large databases. *Proc. of ACM SIGMOD Int. Conference on Management of Data* pages 127-138, 1995.
3. M. Hernandez. A generation of band joins and the merge/purge problem. *Technical report CUCS-005-1995*, Department of Computer Science, Columbia University, 1995.
4. C. Jacquemin and J. Royaute. Retrieving terms and their variants in a lexicalized unification-based framework. *Proc. of the ACM-SIGIR Conference on Research and Development in Information Retrieval* pages 132-141, 1994.
5. A.E. Monge and C.P. Elkan. The field matching problem: Algorithms and applications. *Proc. of the 2nd Int. Conference on Knowledge Discovery and Data Mining* pages 267-270, 1996.
6. A. Siberschatz, M. Stonebraker, and J.D. Ullman. Database research: achievements and opportunities into the 21st century. A report of an NSF workshop on the future of database research. *SIGMOD RECORD*, March 1996.
7. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology* 147:195-197, 1981.