ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

An Efficient Method to Detect Duplicate Data in Databases

G. Vijendar Reddy
Associate Professor,
Department of IT,
GRIET, Bachupally.

N. Siva Lakshmi
M.Tech (SE),
Department of IT,
GRIET, Bachupally.

ABSTRACT:

Duplicate detection is the process of identifying multiple representations of same real world entities.

Today, duplicate detection methods need to process ever larger datasets in ever shorter time: maintaining the quality of a dataset becomes increasingly difficult. We present two novel, progressive duplicate detection algorithms that significantly increase the efficiency of finding duplicates if the execution time is limited: They maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches.

Comprehensive experiments show that our progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work.

Keywords: ***Data cleaning, Duplicate detection, Entity Resolution, Progressiveness***

INTRODUCTION

Data are among the most important assets of a company. But due to data changes and sloppy data entry, errors such as duplicate entries might occur, making data cleansing and in particular duplicate detection indispensable. However, the pure size of today's datasets renders duplicate detection processes expensive. Online retailers, for example, offer huge catalogs comprising a constantly growing set of items from many different suppliers. As independent persons change the product portfolio, duplicates arise. Although there is an obvious need for deduplication, online shops without downtime cannot afford traditional deduplication. Progressive duplicate detection identifies most duplicate pairs early in the detection process. Instead of reducing the overall time needed to finish the entire process, progressive approaches try to reduce the average time after which a duplicate is found.

METHODS AND MATERIAL

Detection of Duplicate Record

The problem of identifying approximately duplicate record in database is an essential step for data cleaning & data integration process. A dynamic web page is displayed to show the results as well as other relevant advertisements that seem relevant to the query. The real world entities have two or more representation in databases. When dealing with large amount of data it is important that there be a well defined and tested mechanism to filter out duplicate result. This keeps the result relevant to the queries. Duplicate record exists in the query result of many web databases especially when the duplicates are defined based on only some 21of the fields in a record. Using exact matching technique Records that are exactly same can be detected. The system that helps user to integrate and compares the query results returned from multiple web databases matches the different sources records that referred to the same real world entity. In this project, we analyze the literature on duplicate record detection.

We cover similarity metrics which are commonly used to detect similar field entries, and present an extensive set of duplicate detection algorithms that can detect approximately duplicate records in a database also the techniques for improving the efficiency and scalability of approximate duplicate detection algorithms are covered. We conclude with coverage of existing tools and with a brief discussion of the big open problems in the area.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

A Generalization of Blocking and Windowing Algorithms for Duplicate Detection

Duplicate detection is the process of finding multiple records in a dataset that represent the same real-world entity. Due to the enormous costs of an exhaustive comparison, typical algorithms select only promising record pairs for comparison. Two competing approaches are blocking and windowing. Blocking methods partition records into disjoint subsets, while windowing methods, in particular the Sorted Neighborhood Method, slide a window over the sorted records and compare records only within the window. We present a new algorithm called Sorted Blocks in several variants, which generalizes both approaches. To evaluate Sorted Blocks, we have conducted extensive experiments with different datasets. These show that our new algorithm needs fewer comparisons to find the same number of duplicates.

Creating Probabilistic Databases from Duplicated Data

A major source of uncertainty in databases is the presence of duplicate items, i.e., records that refer to the same real world entity. However, accurate deduplication is a difficult task and imperfect data cleaning may result in loss of valuable information. A reasonable alternative approach is to keep duplicates when the correct cleaning strategy is not certain, and utilize an efficient probabilistic query answering technique to return query results along with probabilities of each answer being correct. In this project, we present a flexible modular framework for scalably creating a probabilistic database out of a dirty relation of duplicated data and overview the challenges raised in utilizing this framework for large relations of string data. We study the problem of associating probabilities with duplicates that are detected using state-of-the-art scalable approximate join methods. We argue that standard thresholding techniques are not sufficiently robust for this task, and propose new clustering algorithms suitable for inferring duplicates and their associated probabilities. We show that the inferred probabilities accurately reflect the error in duplicate records.

Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem

The problem of merging multiple databases of information about common entities is frequently encountered in KDD and decision support applications in large commercial and government organizations. The problem we study is often called the Merge/Purge problem and is difficult to solve both in scale and accuracy. Large repositories of data typically have numerous duplicate information entries about the same entities that are difficult to cull together without an intelligent "equational theory" that identifies equivalent items by a complex, domain-dependent matching process. We have developed a system for accomplishing this data cleansing task and demonstrate its use for cleansing lists of names of potential customers in a direct marketing-type application. Our results for statistically generated data are shown to be accurate and effective when processing the data multiple times using different keys for sorting on each successive pass. Combing results of individual passes using transitive closure over the independent results, produces far more accurate results at lower cost.

The system provides a rule programming module that is easy to program and quite good at finding duplicates especially in an environment with massive amounts of data. This project details improvements in our system, and reports on the successful implementation for a real-world database that conclusively validates our results previously achieved for statistically generated data.

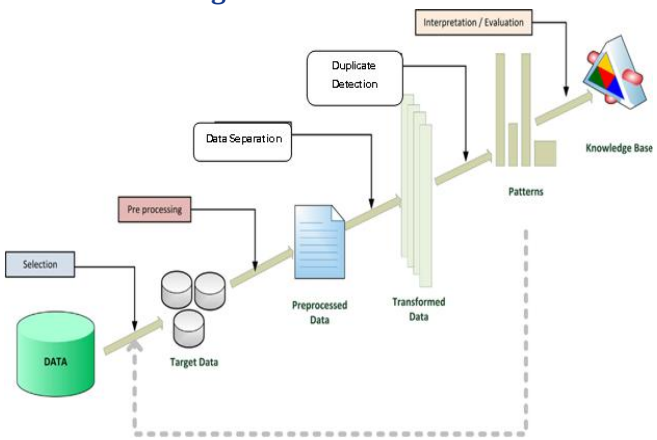
A survey of indexing techniques for scalable record Linkage and Deduplication

Record linkage is the process of matching records from several databases that refer to the same entities. When applied on a single database, this process is known as deduplication. Increasingly, matched data are becoming important in many application areas, because they can contain information that is not available otherwise, or that is too costly to acquire.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Removing duplicate records in a single database is a crucial step in the data cleaning process, because duplicates can severely influence the outcomes of any subsequent data processing or data mining. With the increasing size of today's databases, the complexity of the matching process becomes one of the major challenges for record linkage and deduplication. In recent years, various indexing techniques have been developed for record linkage and deduplication. They are aimed at reducing the number of record pairs to be compared in the matching process by removing obvious nonmatching pairs, while at the same time maintaining high matching quality. This project presents a survey of 12 variations of 6 indexing techniques. Their complexity is analyzed, and their performance and scalability is evaluated within an experimental framework using both synthetic and real data sets. No such detailed survey has so far been published.

Architecture Diagram



Dataset Collection

To collect and/or retrieve data about activities, results, context and other factors. It is important to consider the type of information it want to gather from your participants and the ways you will analyze that information. The data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable. After collecting the data to store the Database.

Preprocessing Method

Data preprocessing or Data cleaning, Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data. And also used to removing the unwanted data. Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user.

Data Separation

After completing the pre-processing, the data separation to be performed. The blocking algorithms assign each record to a fixed group of similar records (the blocks) and then compare all pairs of records within these groups. Each block within the block comparison matrix represents the comparisons of all records in one block with all records in another block, the equidistant blocking; all blocks have the same size.

Duplicate Detection

The duplicate detection rules set by the administrator, the system alerts the user about potential duplicates when the user tries to create new records or update existing records. To maintain data quality, you can schedule a duplicate detection job to check for duplicates for all records that match a certain criteria. You can clean the data by deleting, deactivating, or merging the duplicates reported by a duplicate detection.

Quality Measures

The quality of these systems is, hence, measured using a cost-benefit calculation. Especially for traditional duplicate detection processes, it is difficult to meet a budget limitation, because their runtime is hard to predict. By delivering as many duplicates as possible in a given amount of time, progressive processes optimize the cost-benefit ratio. In manufacturing, a measure of excellence or a state of being free from defects, deficiencies and significant variations. It is brought about by strict and consistent commitment to

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

certain standards that achieve uniformity of product in order to satisfy specific customer or user requirements.

Proposed System

In this work, however, we focus on progressive algorithms, which try to report most matches early on, while possibly slightly increasing their overall runtime.

To achieve this, they need to estimate the similarity of all comparison candidates in order to compare most promising record pairs first.We propose two novel, progressive duplicate detection algorithms namely progressive sorted neighborhood method (PSNM), which performs best on small and almost clean datasets, and progressive blocking (PB), which performs best on large and very dirty datasets. Both enhance the efficiency of duplicate detection even on very large datasets. We propose two dynamic progressive duplicate detection algorithms, PSNM and PB, which expose different strengths and outperform current approaches.

We introduce a concurrent progressive approach for the multi-pass method and adapt an incremental transitive closure algorithm that together forms the first complete progressive duplicate detection workflow. We define a novel quality measure for progressive duplicate detection to objectively rank the performance of different approaches. We exhaustively evaluate on several real-world datasets testing our own and previous algorithms.

Advantages of Proposed System

Improved early quality and same eventual quality and our algorithms PSNM and PB dynamically adjust their behavior by automatically choosing optimal parameters, e.g., window sizes, block sizes, and sorting keys, rendering their manual specification superfluous.

In this way, we significantly ease the parameterization complexity for duplicate detection in general and contribute to the development of more user interactive applications.

Proposed Algorithm Progressive SNM

The algorithm takes five input parameters: D is a reference to the data, which has not been loaded from disk yet. The sorting key K defines the attribute or attributes combination that should be used in the sorting step. W specifies the maximum window size, which corresponds to the window size of the traditional sorted neighborhood method. When using early termination, this parameter can be set to an optimistically high default value. Parameter I defines the enlargement interval for the progressive iterations. For now, assume it has the default value 1. The last parameter N specifies the number of records in the dataset. This number can be gleaned in the sorting step, but we list it as a parameter for presentation purposes. Progressive Sorted Neighborhood Require: dataset reference D, sorting key K, window size W, enlargement interval size I, number of records N

Algorithm 1. Progressive Sorted Neighborhood

```
Require: dataset reference D, sorting key K, window size
    W, enlargement interval size I, number of records N
 1: procedure PSNM(D, K, W, I, N)
 2:     pSize ← calcPartitionSize(D)
 3:     pNum ← ⌈N/(pSize − W + 1)⌉
 4:     array order size N as Integer
 5:     array recs size pSize as Record
 6:     order ← sortProgressive(D, K, I, pSize, pNum)
 7:     for currentI ← 2 to ⌈W/I⌉ do
 8:         for currentP ← 1 to pNum do
 9:             recs ← loadPartition(D, currentP)
10:             for dist ∈ range(currentI, I, W) do
11:                 for i ← 0 to |recs| − dist do
12:                     pair ← ⟨recs[i], recs[i + dist]⟩
13:                     if compare(pair) then
14:                         emit(pair)
15:                         lookAhead(pair)
```

Progressive Blocking

The algorithm accepts five input parameters: The dataset reference D specifies the dataset to be cleaned and the key attribute or key attribute combination K defines the sorting. The parameter R limits the maximum block range, which is the maximum rank-distance of two blocks in a block pair, and S specifies the size of the blocks. Finally, N is the size of the input dataset.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Progressive Blocking Require: dataset reference D, key attribute K, maximum block range R, block size S and record number N

Algorithm 2. Progressive Blocking

```
Require: dataset reference D, key attribute K, maximum
    block range R, block size S and record number N
 1: procedure PB(D, K, R, S, N)
 2:     pSize ← calcPartitionSize(D)
 3:     bPerP ← ⌊pSize/S⌋
 4:     bNum ← ⌈N/S⌉
 5:     pNum ← ⌈bNum/bPerP⌉
 6:     array order size N as Integer
 7:     array blocks size bPerP as ⟨Integer, Record[ ]⟩
 8:     priority queue bPairs as ⟨Integer, Integer, Integer⟩
 9:     bPairs ← {⟨1, 1, _⟩, ... ,⟨bNum, bNum, _⟩}
10:     order ← sortProgressive(D, K, S, bPerP, bPairs)
11:     for i ← 0 to pNum − 1 do
12:         pBPs ← get(bPairs, i · bPerP, (i + 1) · bPerP)
13:         blocks ← loadBlocks(pBPs, S, order)
14:         compare(blocks, pBPs, order)
15:     while bPairs is not empty do
16:         pBPs ← {}
17:         bestBPs ← takeBest(⌊bPerP/4⌋, bPairs, R)
18:         for bestBP ∈ bestBPs do
19:             if bestBP[1] − bestBP[0] < R then
20:                 pBPs ← pBPs ∪ extend(bestBP)
21:         blocks ← loadBlocks(pBPs, S, order)
22:         compare(blocks, pBPs, order)
23:         bPairs ← bPairs ∪ pBPs
24: procedure compare(blocks, pBPs, order)
25:     for pBP ∈ pBPs do
26:         ⟨dPairs, cNum⟩ ← comp(pBP, blocks, order)
27:         emit(dPairs)
28:         pBP[2] ← |dPairs| / cNum
```

CONCLUSION

This work introduced the progressive sorted neighborhood method and progressive blocking. Both algorithms increase the efficiency of duplicate detection for situations with limited execution time; they dynamically change the ranking of comparison candidates based on intermediate results to execute promising comparisons first and less promising comparisons later. To determine the performance gain of our algorithms, we proposed a novel quality measure for progressiveness that integrates seamlessly with existing measures. Using this measure, experiments showed that our approaches outperform the traditional SNM by up to 100 percent and related work by up to 30 percent. For the construction of a fully progressive duplicate detection workflow, we proposed a progressive sorting method, Magpie, a progressive multi-pass execution model, Attribute Concurrency, and an incremental transitive closure algorithm. The adaptations AC-PSNM and AC-PB use multiple sort keys concurrently to interleave their progressive iterations. By analyzing intermediate results, both approaches dynamically rank the different sort keys at runtime, drastically easing the key selection problem.

In future work, we want to combine our progressive approaches with scalable approaches for duplicate detection to deliver results even faster. In particular, Kolb et al. introduced a two phase parallel SNM [21], which executes a traditional SNM on balanced, overlapping partitions. Here, we can instead use our PSNM to progressively find duplicates in parallel.

REFERENCES

[1] Thorsten Papenbrock, Arvid Heise, and Felix Naumann "Progressive Duplicate Detection" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 27, NO. 5, MAY 2015

[2] 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.

[3] F. Naumann and M. Herschel, An Introduction to Duplicate Detection. San Rafael, CA, USA: Morgan & Claypool, 2010.

[4] H. B. Newcombe and J. M. Kennedy, "Record linkage: Making maximum use of the discriminating power of identifying information," Commun. ACM, vol. 5, no. 11, pp. 563–566,1962.

[5] M. A. Hernandez and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," Data Mining Knowl. Discovery, vol. 2, no. 1, pp. 9–37, 1998.

[6] X. Dong, A. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in Proc. Int. Conf. Manage. Data, 2005, pp. 85–96.

[7] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller, "Framework for evaluating clustering algorithms in duplicate detection," Proc. Very Large Databases Endowment, vol. 2, pp. 1282– 1293, 2009.

[8] O. Hassanzadeh and R. J. Miller, "Creating probabilistic databases from duplicated data," VLDB J., vol. 18, no. 5, pp. 1141–1166, 2009.

[9] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg, "Adaptive windows for duplicate detection," in Proc. IEEE 28th Int. Conf. Data Eng., 2012, pp. 1073–1083.

[10] S. Yan, D. Lee, M.-Y. Kan, and L. C. Giles, "Adaptive sorted neighborhood methods for efficient record linkage," in Proc. 7th ACM/ IEEE Joint Int. Conf. Digit. Libraries, 2007, pp. 185– 194.

[11] J. Madhavan, S. R. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy, "Web-scale data integration: You can only afford to pay as you go," in Proc. Conf. Innovative Data Syst. Res., 2007.

[12] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in Proc. Int. Conf. Manage. Data, 2008, pp. 847–860.

[13] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k set similarity joins," in Proc. IEEE Int. Conf. Data Eng., 2009, pp. 916–927.

[14] P. Indyk, "A small approximately min-wise independent family of hash functions," in Proc. 10th Annu. ACM-SIAM Symp. Discrete Algorithms, 1999, pp. 454–456.

[15] U. Draisbach and F. Naumann, "A generalization of blocking and windowing algorithms for duplicate detection," in Proc. Int. Conf. Data Knowl. Eng., 2011, pp. 18–24.

[16] H. S. Warren, Jr., "A modification of Warshall's algorithm for the transitive closure of binary relations," Commun. ACM, vol. 18, no. 4, pp. 218–220, 1975.

[17] M. Wallace and S. Kollias, "Computationally efficient incremental transitive closure of sparse fuzzy binary relations," in Proc. IEEE Int. Conf. Fuzzy Syst., 2004, pp. 1561–1565.

[18] F. J. Damerau, "A technique for computer detection and correction of spelling errors," Commun. ACM, vol. 7, no. 3, pp. 171–176, 1964.

[19] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," IEEE Trans. Knowl. Data Eng., vol. 24, no. 9, pp. 1537–1555, Sep. 2012.

[20] B. Kille, F. Hopfgartner, T. Brodt, and T. Heintz, "The Plista dataset," in Proc. Int. Workshop Challenge News Recommender Syst., 2013, pp. 16–23.