International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Progressive Detection of Duplicate Data

Deepa Bhattacharya[1], Sapna Patle[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Ballarpur Institute of Technology, Bamni, Disnt. Chadrapur, Gondwana University
[2]Department of Computer Science and Engineering, Ballarpur Institute of Technology, Bamni Dist. Chandrapur, Gondwana University

Abstract: ***Data duplicate detection is the process of identifying multiple representations of same or real world entities. Nowadays, data duplicate detection methods are needed to process larger datasets in shorter time: maintaining the quality of the datasets and also the entities duplicated becomes increasingly difficult. This application focus on the duplicates in hierarchical data's like XML file. The data can be detected using the detection methods. Here the datasets are loaded in the applications and the processing, extraction, cleaning, separation and detection are carried out to remove the duplicated data. Comprehensive experiments show that our progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work.***

Keywords: Duplicate detection, entity resolution, progressiveness, and data cleaning

1. Introduction

Data are among the most important assets of a company. But due to data changes and bad data entry, errors such as duplicate entries might occur, making data cleansing and in particular duplicate detection indispensable. Thus, the pure size of data renders duplicate detection processes expensive. Many industries and system s depends on the accurate datasets to carry out operations. Online retailers, for example, offer huge catalogues 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 reduplication, online shops without downtime cannot afford traditional reduplication.

Progressive duplicate detection identifies most duplicate data in the detection process. Instead of reducing the overall time that is needed to finish the entire process, progressive approaches try to reduce the average time after which a duplicate is found. In this work

Here, we have two methods to improve the efficiency and finding duplicates data. Initially we use the method progressive sorted neighborhood method (PSNM). It uses the progressive duplicate detection algorithms. In this we sort the input data using predefined sorting key and only compare the records in sorted order. The second method is progressive blocking (PB). It process large and very dirty datasets. It mainly satisfies the two conditions; first one is improved early quality next is same eventual quality. So, both exchange the efficiency of duplicate detection even on very large datasets.

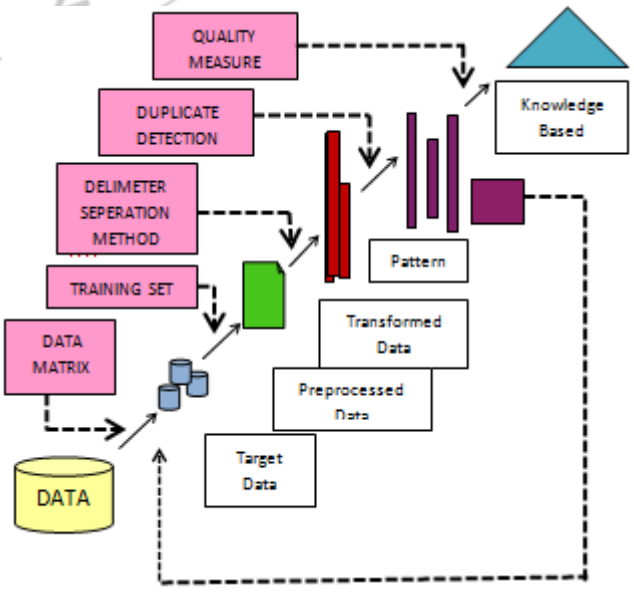
Improved early quality
Let t be an arbitrary target time at which results are needed. Then the progressive algorithm discovers more duplicate pairs at t than the corresponding traditional algorithm. Typically, t is smaller than the overall runtime of the traditional algorithm.

Same eventual quality: If both a traditional algorithm and its progressive version finish execution, without early termination at t, they produce the same results.

2. Related Works

Databases play an important role in today‟s IT-based economy. Many industries and systems depend on the accuracy of databases to carry out operations. Therefore, the quality of the information (or the lack thereof) stored in the databases can have significant cost implications to a system that relies on information to function and conduct business. Much research on duplicate detection, also known as entity resolution and by many other names focuses on pair selection algorithms that try to maximize recall on the one hand and efficiency on the other hand. Adaptive techniques are capable of estimating the quality of comparison candidates. The algorithms use this information to choose the comparison candidates more carefully. In the last few years, the economic need for progressive algorithms also initiated some concrete studies in this domain. For instance, pay-as-you-go algorithms for information integration on large scale datasets have been presented. Other works introduced progressive data cleansing algorithms for the analysis of sensor data streams. However, these approaches cannot be applied to duplicate detection.

2.1 Architecture Diagram



2.2 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.

Pre-processing Method
Data pre-processing 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 pre-processing 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 locking; all blocks have the same size.

Duplicate Detection
The duplicate detection rules set by the administrator, the system alerts the user about potential duplicateswhen 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 certain standards that achieve uniformity of product in order to satisfy specific customer or user requirements.

3. 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. Same eventual quality. Our algorithms PSNM and PB dynamically adjust their behaviour 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.

4. Module Description

The following modules are present in the project.
1. Record addition
2. Attribute selection for duplication finding
3. Input parameter settings for psnm/pb.
4. Progressive sorted neighborhood method algorithm
5. Progressive blocking
6. Proposed system (psnm and progressive blocking)

1. Record addition
In this module, the records are added for the given columns (Employees/Attendance). The records may contain any data. The records are saved in „Employees“ and „Attendance“ table.

2. Attribute selection for duplication finding
In this module, which columns are selected for finding duplicates in records.

3. Input parameter settings for psnm/pb Input parameters (D, K, W, I, N)
In this module, input for the Algorithm PSNM is selected. 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. N is the number of records.

Input parameters (D, K, R, S, N)
In this module, input for the Algorithm PB is selected. 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. R specifies the maximum block range, S Block Size and N Total No. of Records. When using early termination, this parameter can be set to an optimistically high default value.

5. Progressive Sorted Neighborhood Method Algorithm

The PSNM algorithm calculates an appropriate partition size pSize, i.e., the maximum number of records that fit in memory, using the pessimistic sampling function calcPartitionSize(D) in Line 2: If the data is read from a database, the function can calculate the size of a record from the data types and match this to the available main memory. Otherwise, it takes a sample of records and estimates the size of a record with the largest values for each field. The algorithm calculates the number of necessary partitions pNum, while considering a partition overlap of W - 1 records to slide the window across their boundaries. Line 4 defines the order-array, which stores the order of records with regard to the given key K. By storing only record IDs in this array, we assume that it can be kept in memory. To hold the actual records of a current partition, PSNM declares the recs-array.

Algorithm 1.Progresive Sorted Neighbourhood
Require: dataset reference D, sorting key K, window size W, enlargement interval size I, number of record N
1:**procedure**PSNM(D,K,W,I,N)
2:pSize← calcPartationSize(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**current←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** campare(pair)**then**
14: emit(pair)
15: lookAhead(pair)

6. Progressive blocking

In this module, dataset references D, key attribute K, maximum block range R, block size S and record number N are given as input. 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. At first, PB calculates the number of records per partition pSize by using a pessimistic sampling function in Line 2. The algorithm also calculates the number of loadable blocks per partition bPerP, the total number of blocks bNum, and the total number of partitions pNum

Algorithm2: 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 U extend(bestBP)
21: blocks ← loadBlocks(pBPs, S , order)
22: compare(blocks, pBPs, order)
23: bPairs ← bPairs U 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

Reference

[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] S.ramya and C. palaninehru ,” A Study of Progressive Techniques for Efficient Duplicate Detection ” International Journal of Advanced Research in Computer Science and Software Engineering , Volume 5, Issue 11, November 2015.
[3] Dr.M.Mayilvaganan, M.Saipriyanka, “Efficient and Effective Duplicate Detection Evaluating Multiple Data using Genetic Algorithm” International Journal of Innovative Research in Computer and Communication Engineering,Vol. 3, Issue 9, September 2015.
[4] www.ijarcsse.com
[5] S. E. Whang, D. Marmaros, and H. Garcia-Molina, “Pay-as-you-go entity resolution,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 5, pp. 1111–1124, May 2012
[6] 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.