Middle-East Journal of Scientific Research 24 (Techniques and Algorithms in Emerging Technologies): 291-296, 2016
ISSN 1990-9233;

DOI: 10.5829/idosi.mejsr.2016.24.TAET23540

An Efficient Duplicate Detection Based on Navie Block Detection Algorithm

[1]S. Ramya and [2]C. Palani Nehru

[1]PG Scholar, Department of CSE, Vivekanandha College of Engineering for Women, Tamilnadu, India
[2]Assistant Professor, Department of CSE, Vivekanandha College of Engineering for Women, Tamilnadu, India

Abstract: The problem of identifying in the order of duplicate records in databases is an essential step for data cleaning and data integration methods. Most existing advances have relied on generic or manually tuned distance metrics for estimating the similarity of potential duplicates. A variety of experimental methodologies have been used to evaluate the accuracy of duplicate-detection systems. In this paper propose naive detection algorithm by making intelligent guesses which records have a high possibility of representing the same real-world entity, the search space is reduced. An implement naive approach to be used as a baseline simply generates all possible pairs of objects that are stored within the data source. Experiments show that the naive algorithm is better than progressive duplicate detection and that the comprehensive algorithm to some extent improves upon it in terms of efficiency (detected duplicates vs. overall number of comparisons).

Key words: Duplicate detection · Data cleaning · Naive detection · Progressiveness

INTRODUCTION

Duplicate detection is the problem of influential that two different database entries in reality represent the same real-world entity and performing this detection for all objects corresponded to in the database. “Duplicate detection" is also known as record relation, Entity classification, record matching and many other terms. It is a much researched problem with high importance in the areas of master data management, data warehousing and ETL, customer relationship management and data integration. Duplicate detection must solve two intrinsic difficulties: Speedy discovery of all duplicates in large data sets (competence) and correct identification of duplicates and non-duplicates (effectiveness).

Data cleaning is a significant element for developing effective business intelligence applications. The inability to make sure data quality can unconstructively affect downstream data analysis and ultimately key business decisions. A very vital data cleaning operation is that of classifying records which match the same real world entity. For example, due to various errors in data and to differences in gathering of representing data, product names in sales records may not match exactly with records in master product index tables. In these situations, it would be desirable to match similar records transversely relations. This difficulty of matching similar records has been studied in the context of record linkage (e.g. [1-3]) and of identifying approximate duplicate entities in databases.

Several other practical issues arise when training adaptive duplicate-detection systems using machine learning. These include how to competently collect effective training data for the system and how to correctly measure overview accuracy. We can imagine two different scenarios in which machine learning can be used to recover duplicate detection. In the first scenario, the goal is to use machine learning to develop a general duplicate-detection system modified to a specific type of data, such as mailing addresses or bibliographic citations, but not modified to a specific database. In this approach, the concluding databases to be cleaned are not obtainable during the training phrase. We call this the

Corresponding Author: S. Ramya, PG Scholar, Department of CSE, Vivekanandha College of Engineering for Women, Tamilnadu, India.

"shrink-wrap" scenario, since the goal is to develop and market a fixed "shrink-wrapped" software system that any user can be relevant to their own database without further training. In the second scenario, the goal is to train a system to clean a specific database and sample duplicate and non-duplicate pairs from this database be capable of be recognized by the user during the training phase. We call this the "consulting" position, since it seems most suitable under a business model where a company is employing to clean specific databases and trains the software predominantly for each database.

In this paper he naive duplicate detection algorithm simply generates all possible pairs out of the data, but discards reflexive pairs (i.e. a pair that looks like [a, a] containing two references of the same object instance) and symmetric pairs (i.e. [b, a] will be discarded, if [a, b] was already generated).

Related Work: Much research on duplicate detection [4], [5], also known as entity resolution and by many other names centers on pair selection algorithms that try to maximize recollect on the one hand and effectiveness on the other hand. The most important algorithms in this area are Blocking [6] and the sorted neighborhood method (SNM) [7]. Adaptive techniques. Preceding publications on duplicate detection often focus on reducing the overall runtime. Thereby, some of the proposed algorithms are already proficient of approximation the quality of comparison candidates [7]. The algorithms use this information to choose the evaluation candidates more carefully. For the same reason, other advances exploit adaptive windowing techniques, which enthusiastically adjust the window size depending on the amount of recently found duplicates [8], [9]. These adaptive techniques dynamically recover the effectiveness of duplicate detection, but in distinction to our progressive techniques, they need to run for assured stages of time and cannot maximize the efficiency for any given time slot.

Progressive techniques. In the last few years, the profitable need for progressive algorithms also commenced some definite studies in this domain. For occurrence, pay-as-you-go algorithms for information combination on large scale datasets have been presented [10]. Other works introduced progressive data cleansing algorithms for the analysis of sensor data streams [11]. However, these advances cannot be applied to duplicate detection.

Xiao *et al.* proposed a top-k similarity join that uses a particular index structure to estimation capable evaluation candidates [11]. This approach progressively determines duplicates and also relieves the parameterization problem. The focus differs: Xiao *et al.* find the top-k most similar duplicates regardless of how long this takes by weakening the similarity threshold; we find as many duplicates as possible in a given time. That these reproductions are also the most similar ones is a side effect of our approaches.

Pay-As-You-Go Entity Resolution established three kinds of progressive duplicate detection techniques, called "hints". A hint defines a perhaps good implementation order for the evaluations in order to match capable record pairs earlier than less capable record pairs. However, all presented hints produce fixed orders for the comparisons and neglect the chance to dynamically regulate the comparison order at runtime based on intermediate results. Some of our techniques directly tackle this issue. Furthermore, the presented duplicate detection advances calculate a hint only for a specific separation, which is a (perhaps large) subset of records that fits into main memory. By implementation one separation of a huge dataset after another, the overall duplicate detection process is no longer progressive. This issue is only partly addressed, which proposes to calculate the hints using all partitions. The algorithms presented in our paper use a inclusive ranking for the comparisons and consider the limited amount of available main memory. The third issue of the algorithms established by Whang *et al.* relates to the planned pre-partitioning strategy: By using minhash signatures [12] for the partitioning, the partitions do not be related. However, such an overlap improves the pair-selection [13] and thus our algorithms consider not be separating blocks as well. In distinction, we also progressively solve the multi-pass method and transitive closure computation, which are critical for a completely progressive workflow. Finally, we provide a more general estimate on considerably larger datasets and employ a novel quality measure to quantify the presentation of our progressive algorithms.

Additive techniques. By merging the sorted neighborhood method with blocking techniques, pair-selection algorithms can be built that prefers the comparison candidates much more precisely. The Sorted Blocks algorithm [13], for instance, applies blocking techniques on a set of input records and then slides a small window between the different blocks to select further comparison candidates.

Proposed Work: In this section the proposed algorithm naive block partitioning are responsible for selecting pairs of records from the extractors that should be classified as duplicate or non-duplicate. The naive supports that follow

a pair-wise comparison pattern and it already provides a increasing selection of such algorithms. An implemented naive approach to be used as a baseline generates all possible pairs of Entities that are stored within the data source(s). Each pair is returned only once. So if (a, b) is already returned, (b, a) is not. This algorithm use preprocessing, such as sorting for the Sorted Neighborhood Process or partitioning for the Blocking Method. Therefore, each algorithm can execute a preprocessing step before returning record pairs. In case of sorting, naive allows the definition of a sorting key. A sorting key collects a list of different sub-keys which specify attributes or part of attribute values. The sorting can be executed by an in-memory (for small datasets) or by label based sorter.

Data Extractors: The data extractor element is used to extract data from any data source that is sustained by the toolkit and to alter the data into the internal JSON format. Currently, For each data extractor, a record identifier, consisting of one or many attributes, can be defined and furthermore a global ID is assigned to each data extractor, which is also saved within the removed records. This allows a evaluation of records from different sources without the essential of an extractor-wide unique identifier.

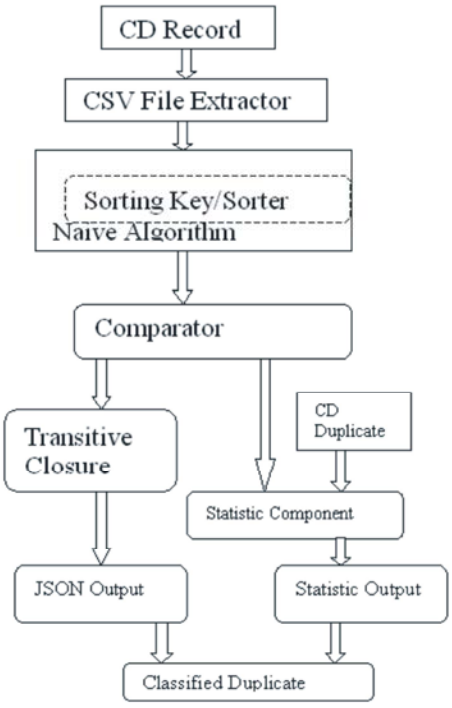

Fig. 1: Naive Detection Architecture

Preprocessor: The preprocessor is used to gather statistics while extracting the data, e.g., counting the number of records or (dissimilar) values. After the extraction phase, each preprocessor instance is available within the algorithm and might be used within comparators that need preprocessing information.

Partitioning Algorithms: In this proposed system two admired families of methods for duplicate detection. Blocking methods partition data into numerous blocks or partitions and compare only tuples within a partition. Windowing methods on the other hand sort the data, slide a window transversely the arranged data and compare only within the window.

Blocking: Blocking methods practice the simple idea of separating the set of tuples into disjoint partitions (blocks) and then evaluating all pairs of tuples only within each block. Thus, the overall number of comparisons is greatly reduced; see Tab. 1 for an overview of the computational difficulty of the different methods compared to the extensive approach of comparing all the pairs of tuples. An important decision for the blocking method is the choice of a good partitioning predicate, which establishes the number and size of the partitions. They should be chosen in a manner that possible duplicates appear in the same partition. E.g., for CRM functions a typical partitioning is by zip-code or by the first few digits of zip-codes. If two duplicate tuples have maintained the same zip code, they appear in the same partition and thus can be predictable as duplicates. Other partitioning might be by last name or some fixed-sized pre x of them, by street name, by employer, etc. In broad, partitions of generally same size are preferable. For simplicity we imagine in the following that separations are of equal size. Finally, a transitive closure is created over all detected duplicates, because duplicity is intrinsically a transitive relation and thus more correct duplicate pairs can be reported.

To detect duplicates that change in the partitioning attribute, a multi-pass method is in use for Blocking methods to perform several runs, each time with a different partitioning predicate.

Windowing: Windowing methods are vaguely more detailed than blocking methods. The naive, which is divided into three phases. In the first phase, a sorting key is allocated to each tuple. The key does not have be

distinctive and can be produced by concatenating values (or substrings of values) from dissimilar attributes. In the second phase, all tuples are sorted according to that key. As in the blocking method, the hypothesis is that duplicates have similar keys and are thus close to each other after sorting. The first two phases are similar to the selection of a partitioning predicate in the blocking method. The final phase of naive slides a window of preset size across the sorted list of tuples. All pairs of tuples that appear in the same window are evaluated. The size of the window (typically between 10 and 20) represents the trade-off between efficiency and effectiveness; larger windows yield longer run-times but discover more duplicates.

Duplicate Comparator: The proposed algorithm reducing the number of comparisons by making intelligent guesses as to which pairs of tuples have a chance of being duplicates. Naive rely on some intrinsic orderings of the data and the assumption that tuples that are close to each other with deference to that order have a higher chance of being duplicates than other pairs of tuples.

To further compare naïve analyze the relationship between number of blocks b and window size w. To achieve the same number of evaluations for both methods we calculate:

$$n\left(\frac{n-b}{2b}\right) = (w-1)(n-\frac{w}{2})$$

$$\leftrightarrow b = \frac{n^2}{2wn - n - w^2 + w} \quad (1)$$

with n = 20 and w = 3 confirm that the partitioning into 4 partitions approximately accomplishes the same number of comparisons as the windowing method with window size 3:

Duplicate Detection: Windowing methods and blocking are two extreme examples concerning the overlap of the partitions. Let U be the intersection between two partitions P1 and P2, which we call.

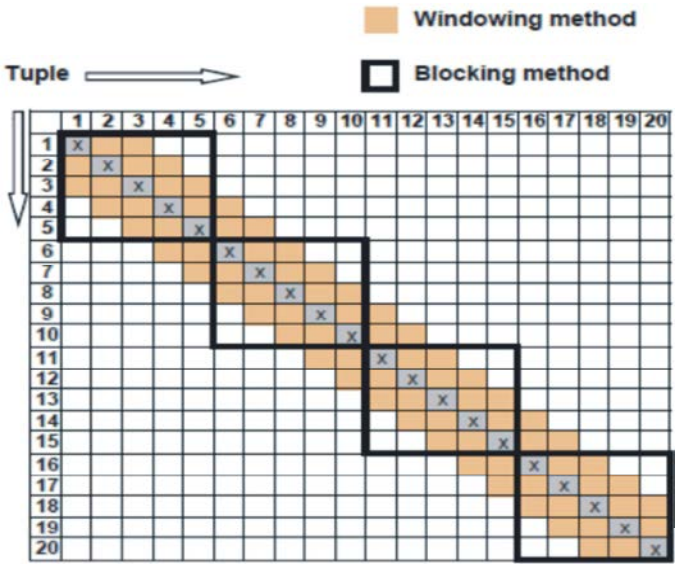

Fig. 2: Comparing blocking and windowing

$$U_{p1,p2} = P1 \cap P2 \quad (2)$$

$$u = |U_{p1,p2}| \quad (3)$$

In the best case when using separating methods, tuples that are really true duplicates are assigned to the same partition. The model be related between two partitions should be big enough, so that real duplicates that for any reason are not in the same partition are recognized. On the other hand, the overlap should not be too high, thus resulting in a large increase of record comparisons. The model be related depends on the data set and has to be determined manually for each use case.

The basic scheme of the new Sorted Blocks method is to first sort all tuples so that duplicates are close in the sort sequence, then separation the records into disjoint sorted sub-sets and finally to overlap the partitions. The size of the be related can be defined using u, e.g.,

u = 3 means that three tuples of each neighboring partition are element of the overlap, which hence has a total size of 2u. Within the overlap, a fixed size window with size u + 1 is slid across the sorted data and all reports within the window are compared. In this way, the additional complexity of the be related is linear. Note that this windowing technique is used only in the overlapping part; within a partition all pairs of tuples are compared.

RESULTS AND DISCUSSION

The most proficient overlap-setting of u = 2 between the partitions is selected experimentally. Compares the results of different overlaps with u = 2 as a baseline. For each F-Measure-value the graphs show the least number of additional comparisons necessary to achieve that value. A first observation is that the graphs are always above zero, so naive Blocks with overlap u > 2 needs more record comparisons than naive Blocks with u = 2 to achieve the same F-Measure and is hence less proficient. Another effect observes is that with an increasing F-Measure, the number of additional comparisons generally decreases. Because the F-Measure depends on the recall and therefore on the number of record evaluations, it can be increased with an increasing partition size, but this reduces the effect of the overlap between the partitions.

The CD-dataset1 contains various records about music and audio CDs. The DBLP-dataset 2 is a bibliographic index on computer science journals and proceedings. In contrast to the other two datasets, DBLP includes many, large groups of similar article representations. The CSX-dataset3 contains bibliographic data used by the CiteSeerX search engine for systematic digital literature. CSX also stores the full abstracts of all its publications in text-format. These conceptual are the largest attributes in our experiments. Our work focuses on increasing efficiency while keeping the same effectiveness. Hence, we assume a given, correct similarity measure; it is treated as an exchangeable black box. For our experiments, however, we use the Levenshtein similarity. This similarity measure achieved an actual precision of 95 percent on the CD-dataset, for which we have a true gold standard.

We first generally evaluate the performance of our advances and compare them to the conventional sorted neighborhood method and the sorted list of record pairs presented. Then, we test our algorithms using a much larger dataset and a concrete use case. The graphs used for performance dimensions plot the total number of description duplicates over time. Each duplicate is a positively matched record pair. For enhanced readability, we manually marked some facts points from the many hundred measured data points that make up a graph.

Table 1: Compare Duplicate vs Time

	Time				
Algorithm	0.1	0.5	1	1.5	2
SNM	50	98	124	134	143
PSNM	65	121	143	186	210
Naïve	78	156	197	243	297

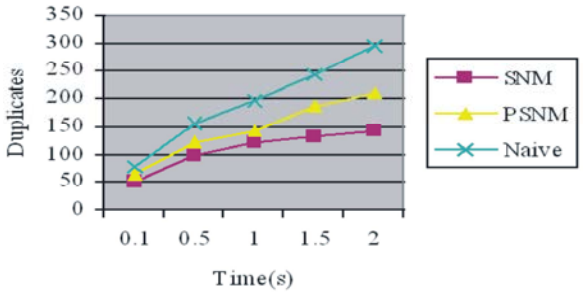

Fig. 3: Performance comparison of the traditional SNM and the PSNM and Naive

CONCLUSION AND FUTURE WORK

Duplicate detection is an essential problem in data cleaning and an adaptive approach that learns to identify duplicate records for a detailed domain has clear advantages over static methods. Experimental results demonstrate that trainable similarity measures are capable of learning the definite notion of similarity that is appropriate for a specific domain. We presented naive and distance measures that improve the duplicate detection. It would be interesting to compare static-active sample selection with dynamic learning techniques, since the results would help determine how much active learning helps select revealing instruction pairs beyond that of just balancing the extremely uneven class distribution in the training data. The proposed method for weakly-labeled negative collection results in training sets that never enclose “near-miss” negative examples, since the inverse blocking method guarantees that records preferred for negative pairs are different. While there were no indications that this methodology hurts accuracy compared to random selection of true negative examples, it would be interesting to evaluate weakly-labeled negative selection to active learning to conclude how

much accuracy enhancement can be attained from employing near-miss negatives during training. Comparing active learning with a combination of the two techniques that we proposed, static-active duplicate selection and weakly-labeled non-duplicate collection could also yield interesting tentative results. Finally, exploring approaches to using weakly-labeled evidence pairs as both duplicate and non-duplicate training patterns could potentially direct to new simply unsupervised duplicate detection methods.

REFERENCES

1. Agichtein, E. and V. Ganti, 2004. Mining reference tables for automatic text segmentation. In Proceedings of ACM SIGKDD.
2. Arasu, A., V. Ganti and R. Kaushik, 2006. Efficient exact set-similarity joins. In Proceedings of VLDB.
3. Argamon-Engelson, S. and I. Dagan, 1999. Committee-based sample selection for probabilistic classifiers. Journal of Artificial Intelligence research.
4. Elmagarmid, A.K., P.G. Ipeirotis and V.S. Verykios, 2007. "Duplicate record detection: A survey," IEEE Trans. Knowl. Data Eng., 19(1): 1-16, Jan. 2007.
5. Naumann, F. and M. Herschel, 2010. An Introduction to Duplicate Detection. San Rafael, CA, USA: Morgan and Claypool.
6. Newcombe, H.B. and J.M. Kennedy, 1962. "Record linkage: Making maximum use of the discriminating power of identifying information," Commun. ACM, 5(11): 563-566.
7. Hern andez, M.A. and S.J. Stolfo, 1998. "Real-world data is dirty: Data cleansing and the merge/purge problem," Data Mining Knowl. Discovery, 2(1): 9-37.
8. Draisbach, U., F. Naumann, S. Szott and O. Wonneberg, 2012. "Adaptive windows for duplicate detection," in Proc. IEEE 28th Int. Conf. Data Eng., pp: 1073-1083.
9. Yan, S., D. Lee, M.Y. Kan and L.C. Giles, 2007. "Adaptive sorted neighborhood methods for efficient record linkage," in Proc. 7th ACM/ IEEE Joint Int. Conf. Digit. Libraries, pp: 185-194.
10. Jeffery, S.R., M.J. Franklin and A.Y. Halevy, 2008. "Pay-as-you-go user feedback for dataspace systems," in Proc. Int. Conf. Manage. Data, pp: 847-860.
11. Xiao, C., W. Wang, X. Lin and H. Shang, 2009. "Top-k set similarity joins," in Proc. IEEE Int. Conf. Data Eng., pp: 916-927.
12. Indyk, P., 1999. "A small approximately min-wise independent family of hash functions," in Proc. 10th Annu. ACM-SIAM Symp. Discrete Algorithms, pp: 454-456.
13. Draisbach, U. and F. Naumann, 2011. "A generalization of blocking and windowing algorithms for duplicate detection," in Proc. Int. Conf. Data Knowl. Eng., pp: 18-24.