On Demand Quality of web services using Ranking by multi criteria

N. Rajanikanth
M.Tech
Dept of CSE
Vivekananda Institute Technology & Science, Karimnagar.
Rajanikanth86@gmail.com

Prof. P.Pradeep Kumar
Hod, CSE Dept
Vivekananda Institute Technology & Science, Karimnagar.
pkpuram@yahoo.com

B. Meena
M.Tech(CSE),Asst. Porfessor
Dept of CSE
Vivekananda Institute Technology & Science, Karimnagar.
vinaymeena@gmail.com

Abstract - In the Web database scenario, the records to match are highly query-dependent, since they can only be obtained through online queries. Moreover, they are only a partial and biased portion of all the data in the source Web databases. Consequently, hand-coding or offline-learning approaches are not appropriate for two reasons. First, the full data set is not available beforehand, and therefore, good representative data for training are hard to obtain. Second, and most importantly, even if good representative data are found and labeled for learning, the rules learned on the representatives of a full data set may not work well on a partial and biased part of that data set.

Keywords: - SOA, Web Services, Networks

Introduction

Today, more and more databases that dynamically generate Web pages in response to user queries are available on the Web. These Web databases compose the deep or hidden Web, which is estimated to contain a much larger amount of high quality, usually structured information and to have a faster growth rate than the static Web. Most Web databases are only accessible via a query interface through which users can submit queries. Once a query is received, the Web server will retrieve the corresponding results from the back-end database and return them to the user.

To build a system that helps users integrate and, more importantly, compare the query results returned from multiple Web databases, a crucial task is to match the different sources' records that refer to the same real-world entity. The problem of identifying duplicates, that is, two (or more) records describing the same entity, has attracted much attention from many research fields, including Databases, Data Mining, Artificial Intelligence, and Natural Language Processing. Most previous work is based on predefined matching rules hand-coded by domain experts or matching rules learned offline by some learning method from a set of training examples. Such approaches work well in a traditional database environment, where all instances of the target databases can be readily accessed, as long as a set of high-quality representative records can be examined by experts or selected for the user to label.

Previous Work

Data integration is the problem of combining information from multiple heterogeneous databases. One step of data integration is relating the primitive objects that appear in the different databases specifically, determining which sets of identifiers refer to the same real-world entities. A number of recent research papers have addressed this problem by exploiting similarities in the textual names used for objects in different databases. (For example one might suspect that two objects from different databases named “USAMA FAYYAD” and “Usama M. Fayyad” ” respectively might refer to the same person.) Integration techniques based on textual similarity are especially useful for databases found on the Web or obtained by extracting information from text, where descriptive names generally exist but global object identifiers are rare. Previous publications in using textual similarity for data integration have considered a number of related tasks. Although the terminology is not completely standardized, in this paper we define *entity-name matching* as the task of taking two lists of entity names from two different sources and determining which pairs of names are co-referent (*i.e.*, refer to the same real-world entity). We define *entity-name clustering* as the task of taking a single list of entity names and assigning entity names to clusters such that all names in a cluster are co-referent. Matching is important in attempting to join information across of pair of relations from different databases, and clustering is important in removing duplicates from a relation that has been drawn from the union of many different information sources. Previous work in this area includes work in distance functions for matching and scalable matching and clustering algorithms. Work in *record linkage* is similar but does not rely as heavily on textual similarities. [1]

Important business decisions; therefore, accuracy of such analysis is crucial. However, data received at the data warehouse from external sources usually contains errors: spelling mistakes, inconsistent conventions, etc. Hence, significant amount of time and money are spent on *data cleaning*, the task of detecting and correcting errors in data. The problem of detecting and eliminating duplicated data is one of the major problems in the broad area of data cleaning and data quality. [2]

Many times, the same logical real world entity may have multiple representations in the data warehouse. For example, when Lisa purchases products from SuperMart twice, she might be entered as two different customers due to data entry errors. Such duplicated information can significantly increase direct mailing costs because several customers like Lisa may be sent multiple catalogs. Moreover, such duplicates can cause incorrect results in analysis queries (say, the number of SuperMart customers in Seattle), and erroneous data mining models to be built. We refer to this problem of detecting and eliminating multiple distinct records representing the same real world entity as the *fuzzy duplicate elimination problem*, which is sometimes also called merge/purge, dedup, record linkage problems. This problem is different from the standard duplicate elimination problem, say for answering "select distinct" queries, in relational database systems which considers two tuples to be duplicates if they match exactly on all attributes. However, data cleaning deals with *fuzzy duplicate elimination*, which is our focus in this paper. Henceforth, we use *duplicate elimination* to mean fuzzy duplicate elimination. [2]

Proposed System

Weighted Component Similarity Summing Classifier

In our algorithm, classifier plays a vital role. At the beginning, it is used to identify some duplicate vectors when there are no positive examples available. Then, after iteration begins, it is used again to cooperate with other classifier to identify new duplicate vectors. Because no duplicate vectors are available initially, classifiers that need class information to train, such as decision tree, cannot be used. An intuitive method to identify duplicate vectors is to assume that two records are duplicates if most of their fields that are under consideration are similar. On the other hand, if all corresponding fields of the two records are dissimilar, it is unlikely that the two records are duplicates. To evaluate the similarity between two records, we combine the values of each component in the similarity vector for the two records. Different fields may have different importance when we decide whether two records are duplicates. The importance is usually data-dependent, which, in turn, depends on the query in the Web database scenario.

Component Weight Assignment

In this classifier, we assign a weight to a component to indicate the importance of its corresponding field. The similarity between two duplicate records should be close to 1. For a duplicate vector that is formed by a pair of duplicate records r1 and r2, we need to assign large weights to the components with large similarity values and small weights to the components with small similarity values. The similarity for two nonduplicate records should be close to 0. Hence, for a nonduplicate vector that is formed by a pair of nonduplicate records r1 and r2, we need to assign small weights to the components with large similarity values and large weights to the components with small similarity values. The component will be assigned a small weight if it usually has a small similarity value in the duplicate vectors.

Duplicate Identification

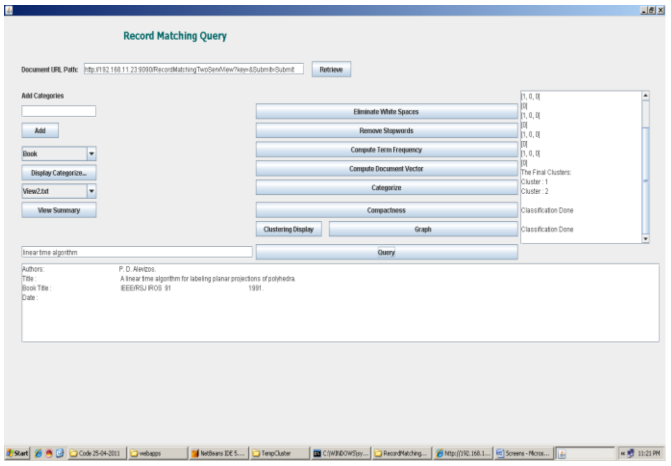
After we assign a weight for each component, the duplicate vector detection is rather intuitive. Two records r1 and r2 are duplicates if they are similar, i.e., if their similarity value is equal to or greater than a similarity threshold. In general, the similarity threshold should be close to 1 to ensure that the identified duplicates are correct. Increasing the value of similarity will reduce the number of duplicate vectors identified while, at the same time, the identified duplicates will be more precise.

Similarity Calculation

The similarity calculation quantifies the similarity between a pair of record fields. As the query results to match are extracted from HTML pages, namely, text files, we only consider string similarity. Given a pair of strings a similarity function calculates the similarity score between Sa and Sb, which must be between 0 and 1. Since the similarity function is orthogonal to the iterative duplicate detection, any kind of similarity calculation method can be employed. Domain knowledge or user preference can also be incorporated into the similarity function. In particular, the similarity function can be learned if training data is available.

Results

The concept of this paper is implemented and different results are shown below



Document URL Path: http://192.168.11.23:9090/RecordMatchingTwoServView?key=&Submit=Submit

Book

View2.txt

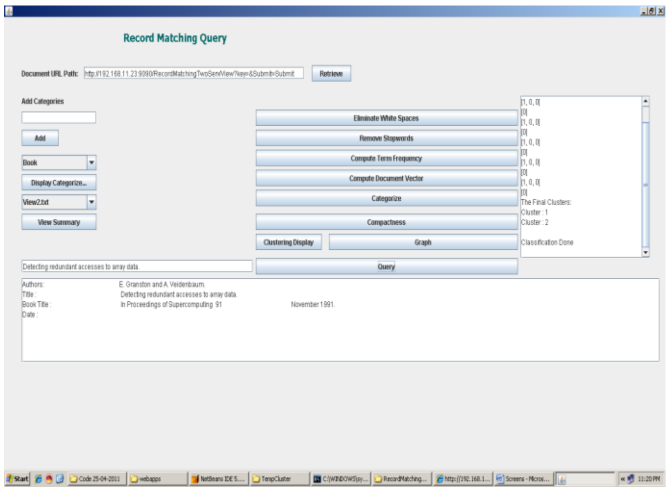
```
[1, 0, 0]
[0]
[1, 0, 0]
[0]
[1, 0, 0]
[0]
[1, 0, 0]
[0]
The Final Clusters:
Cluster : 1
Cluster : 2

Classification Done

Classification Done
```

linear time algorithm

```
Authors:        P. D. Alevizos.
Title :         A linear time algorithm for labeling planar projections of polyhedra.
Book Title :    IEEE/RSJ IROS 91                1991.
Date :
```



Document URL Path: http://192.168.11.23:9090/RecordMatchingTwoServView?key=&Submit=Submit

Book

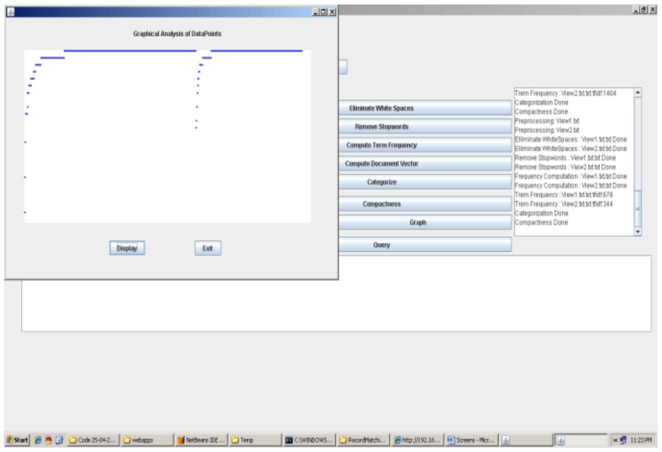
View2.txt

```
[1, 0, 0]
[0]
[1, 0, 0]
[0]
[1, 0, 0]
[0]
[1, 0, 0]
[0]
[1, 0, 0]
[0]
The Final Clusters:
Cluster : 1
Cluster : 2

Classification Done
```

Detecting redundant accesses to array data.

```
Authors:        E. Granston and A. Veidenbaum.
Title :         Detecting redundant accesses to array data.
Book Title :    In Proceedings of Supercomputing 91          November 1991.
Date :
```



```
Trem Frequency : View2.txt.txt tfidf:1404
Categorization Done
Compactness Done
Preprocessing: View1.txt
Preprocessing: View2.txt
Elliminate WhiteSpaces : View1.txt.txt Done
Elliminate WhiteSpaces : View2.txt.txt Done
Remove Stopwords : View1.txt.txt Done
Remove Stopwords : View2.txt.txt Done
Frequency Computation : View1.txt.txt Done
Frequency Computation : View2.txt.txt Done
Trem Frequency : View1.txt.txt tfidf:676
Trem Frequency : View2.txt.txt tfidf:344
Categorization Done
Compactness Done
```

Performance Analysis

The proposed paper is implemented in Java technology on a Pentium-III PC with 20 GB hard-disk and 256 MB RAM with apache web server. The propose paper's concepts shows efficient results and has been efficiently tested on different Messages.

Conclusion

Duplicate detection is an important step in data integration and most state-of-the-art methods are based on offline learning techniques, which require training data. In the Web database scenario, where records to match are greatly query-dependent, a pretrained approach is not applicable as the set of records in each query's results is a biased subset of the full data set. To overcome this problem, we presented an unsupervised, online approach, UDD, for detecting duplicates over the query results of multiple Web databases. Two classifiers, WCSS and SVM, are used cooperatively in the convergence step of record matching to identify the duplicate pairs from all potential duplicate pairs iteratively.

References

[1] W.W. Cohen and J. Richman, "Learning to Match and Cluster Large High-Dimensional Datasets for Data Integration," Proc. ACM SIGKDD, pp. 475-480, 2002.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti, "Eliminating Fuzzy Duplicates in Data Warehouses," Proc. 28th Int'l Conf. Very Large Data Bases, pp. 586-597, 2002.

[3] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and Efficient Fuzzy Match for Online Data Cleaning," Proc. ACM SIGMOD, pp. 313-324, 2003.

[4] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate String Joins in a Database (Almost) for Free," Proc. 27th Int'l Conf. Very Large Data Bases, pp. 491-500, 2001.

[5] X. Dong, A. Halevy, and J. Madhavan, "Reference Reconciliation in Complex Information Spaces," Proc. ACM SIGMOD, pp. 85-96, 2005.

[6] W.W. Cohen, H. Kautz, and D. McAllester, “Hardening Soft Information Sources,” Proc. ACM SIGKDD, pp. 255-259, 2000.

[7] P. Christen, T. Churches, and M. Hegland, “Febrl—A Parallel Open Source Data Linkage System,” Advances in Knowledge Discovery and Data Mining, pp. 638-647, Springer, 2004.

[8] P. Christen and K. Goiser, “Quality and Complexity Measures for Data Linkage and Deduplication,” Quality Measures in Data Mining, F. Guillet and H. Hamilton, eds., vol. 43, pp. 127-151, Springer, 2007.