

Record Matching over Query Results from Multiple Web Databases

Weifeng Su, Jiying Wang, and Frederick H. Lochovsky, *Member, IEEE Computer Society*

Abstract—Record matching, which identifies the records that represent the same real-world entity, is an important step for data integration. Most state-of-the-art record matching methods are supervised, which requires the user to provide training data. These methods are not applicable for the Web database scenario, where the records to match are query results dynamically generated on-the-fly. Such records are query-dependent and a prelearned method using training examples from previous query results may fail on the results of a new query. To address the problem of record matching in the Web database scenario, we present an unsupervised, online record matching method, UDD, which, for a given query, can effectively identify duplicates from the query result records of multiple Web databases. After removal of the same-source duplicates, the “presumed” nonduplicate records from the same source can be used as training examples alleviating the burden of users having to manually label training examples. Starting from the nonduplicate set, we use two cooperating classifiers, a weighted component similarity summing classifier and an SVM classifier, to iteratively identify duplicates in the query results from multiple Web databases. Experimental results show that UDD works well for the Web database scenario where existing supervised methods do not apply.

Index Terms—Record matching, duplicate detection, record linkage, data deduplication, data integration, Web database, query result record, SVM.

1 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. For example, Fig. 1 shows some of the query results returned by two online bookstores, *booksamillion.com* and *abebooks.com*, in response to the same query “Harry Potter” over the Title field. It can be seen that the record numbered 3 in Fig. 1a and the third record in Fig. 1b refer to the same book, since they

have the same ISBN number although their authors differ somewhat. In comparison, the record numbered 5 in Fig. 1a and the second record in Fig. 1b also refer to the same book if we are interested only in the book title and author.¹

The problem of identifying duplicates,² 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.³ Most previous work⁴ 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.

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

- W. Su is with the Computer Science and Technology Program, BNU-HKBU United International College, 28, Jinfeng Road, Tangjiawan, Zhuhai, P.R. China, and the Shenzhen Key Laboratory of Intelligent Media and Speech, PKU-HKUST Shenzhen Hong Kong Institution. E-mail: wfsu@uic.edu.hk.
- J. Wang is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. E-mail: wangjy@cityu.edu.hk.
- F.H. Lochovsky is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: fred@cse.ust.hk.

Manuscript received 7 Dec. 2008; revised 9 Feb. 2009; accepted 22 Mar. 2009; published online 15 Apr. 2009.

Recommended for acceptance by D. Srivastava.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2008-12-0639. Digital Object Identifier no. 10.1109/TKDE.2009.90.


1. A book may have several ISBN numbers due to different editions or different publication formats.


2. We use “duplicate identification” and “record matching” interchangeably when appropriate.


3. The problem has also been referred to as record linkage [3], [32], and [35], database merge/purge problem [20], hardening soft databases [11], record matching [34], deduplication [5], [13], and [28], fuzzy duplicate elimination problem [1] and [7], reference matching [27], reference reconciliation [16], and, more recently, entity resolution [4].


4. See Elmagarmid et al. [17] for a survey on duplicate detection techniques.

(a)

3.  **Harry Potter and the Sorcerer's Stone**
by J. K. Rowling; Mary GrandPré / Paperback / June 1999 / ISBN 059035342X
In Stock: Usually ships within 24 hours.
Add To Cart **SAVE 10%** Retail Price: \$6.99 / Our Price: 6.99
Club Price: \$6.29

4.  **Harry Potter and the Chamber of Secrets**
by J. K. Rowling / Paperback / Aug 2000 / ISBN 0439064872
In Stock: Usually ships within 24 hours.
Add To Cart **SAVE 10%** Retail Price: \$6.99 / Our Price: 6.99
Club Price: \$6.29

5.  **Harry Potter and the Prisoner of Azkaban**
by J. K. Rowling / Paperback / Sept 2000 / ISBN 0439136369
In Stock: Usually ships within 24 hours.
Add To Cart **SAVE 10%** Retail Price: \$7.99 / Our Price: 7.99
Club Price: \$7.19

6.  **Harry Potter and the Prisoner of Azkaban**
by J. K. Rowling / Paperback / May 2004 / ISBN 043965548X
In Stock: Usually ships within 24 hours.
Add To Cart **SAVE 10%** Retail Price: \$7.99 / Our Price: 7.99
Club Price: \$7.19

(b)

1. **Harry Potter and the Goblet of Fire** (ISBN:0439139600)
ROWLING, J. K. / GRANDPRE, MARY (ILT)
Bookseller: Aunties Price: US\$ 5.00 Shipping within U.S.A.:
Bookstore [Convert Currency] US\$ 3.50
(Spokane, WA, U.S.A.) [Rates & Speeds] **Add to Basket**
Book Description: Scholastic. Good Trade Paperback. Bookseller Inventory # 0439139600U1
[Bookseller & Payment Information] [More Books from this Seller]

2. **HARRY POTTER AND THE PRISONER OF AZKABAN** (ISBN:0747546290)
ROWLING, J.K.
Bookseller: Bookbundle.com Inc. Price: US\$ 17.96 Shipping within U.S.A.:
(Southington, CT, U.S.A.) [Convert Currency] US\$ 3.00
[Rates & Speeds] **Add to Basket**
Book Description: BLOOMSBURY PUBLISHING PLC. Book Condition: New.
isbn:0747546290 paperback. Bookseller Inventory # 0747546290
[Bookseller & Payment Information] [More Books from this Seller]

3. **Harry Potter and the Sorcerer's Stone** (ISBN:059035342X)
ROWLING, J. K.
Bookseller: Aunties Price: US\$ 4.00 Shipping within U.S.A.:
Bookstore [Convert Currency] US\$ 3.50
(Spokane, WA, U.S.A.) [Rates & Speeds] **Add to Basket**
Book Description: Scholastic. Very Good Trade Paperback. Bookseller Inventory # 059035342XU1
[Bookseller & Payment Information] [More Books from this Seller]

Fig. 1. Example query results from two Web databases. (a) Query results from booksamillion.com. (b) Query results from abebooks.com.

representatives of a full data set may not work well on a partial and biased part of that data set.

To illustrate this problem, consider a query for books of a specific author, such as “J. K. Rowling.” Depending on how the Web databases process such a query, all the result records for this query may well have only “J. K. Rowling” as the value for the Author field. In this case, the Author field of these records is ineffective for distinguishing the records that should be matched and those that should not. To reduce the influence of such fields in determining which records should match, their weighting should be adjusted to be much lower than the weighting of other fields or even be zero. However, if a matching rule is learned from representatives of the full data set, then it is highly unlikely that a rule to deal with such fields will be discovered. Moreover, for each new query, depending on the results returned, the field weights should probably change too, which makes supervised-learning-based methods even less applicable.

To overcome such problems, we propose a new record matching method *Unsupervised Duplicate Detection*

(UDD) for the specific record matching problem of identifying duplicates among records in query results from multiple Web databases. The key ideas of our method are:

1. We focus on techniques for adjusting the weights of the record fields in calculating the similarity between two records. Two records are considered as duplicates if they are “similar enough” on their fields. As illustrated by the previous example, we believe different fields may need to be assigned different importance weights in an adaptive and dynamic manner.
2. Due to the absence of labeled training examples, we use a sample of universal data consisting of record pairs from different data sources⁵ as an approximation for a negative training set as well as the record pairs from the same data source. We believe, and our experimental results verify, that doing so is reasonable since the proportion of duplicate records in the universal set is usually much smaller than the proportion of nonduplicates.

Employing two classifiers that collaborate in an iterative manner, UDD identifies duplicates as follows: First, each field’s weight is set according to its “relative distance,” i.e., dissimilarity, among records from the approximated negative training set. Then, the first classifier, which utilizes the weights set in the first step, is used to match records from different data sources. Next, with the matched records being a positive set and the nonduplicate records in the negative set, the second classifier further identifies new duplicates. Finally, all the identified duplicates and non-duplicates are used to adjust the field weights set in the first step and a new iteration begins by again employing the first classifier to identify new duplicates. The iteration stops when no new duplicates can be identified.

The contributions of this paper include:

- To our knowledge, this is the first work that studies and solves the online duplicate detection problem for the Web database scenario where query results are generated on-the-fly. In this scenario, the importance of each individual field needs to be considered, which may vary widely from query to query. This makes existing work based on hand-coded rules or offline learning inappropriate.
- To our knowledge, this is also the first work that takes advantage of the dissimilarity among records from the same Web database for record matching. Most existing work requires human-labeled training data (positive, negative, or both), which places a heavy burden on users.
- A machine learning algorithm is proposed to learn only from an approximated negative training set, which may contain some positive examples, i.e., noise. Most existing work learns from a positive example set that contains no noise.

The rest of the paper is organized as follows: In Section 2, related work is reviewed. Section 3 first defines the duplicate

5. The terms *data source* and *Web database* are used interchangeably in this paper.

record matching problem in the Web database context, and then, presents the UDD record matching method. Section 4 validates the UDD method through experiments on some real data sets. Section 5 concludes the paper.

2 RELATED WORK

Most record matching methods adopt a framework that uses two major steps ([17] and [23]):

1. *Identifying a similarity function.* Using training examples (i.e., manually labeled duplicate and nonduplicate records) and a set of predefined basis similarity measures/functions over numeric and/or string fields, a single *composite similarity function* over one pair of records, which is a weighted combination (often linear) of the basis functions, is identified by domain experts [20] or learned by a learning method, such as Expectation-Maximization, decision tree, Bayesian network, or SVM ([5], [12], [32], and [35]).
2. *Matching records.* The composite similarity function is used to calculate the similarity between the candidate pairs and highly similar pairs are matched and identified as referring to the same entity.

An important aspect of duplicate detection is to reduce the number of record pair comparisons. Several methods have been proposed for this purpose including standard blocking [21], sorted neighborhood method [20], Bigram Indexing [9], and record clustering ([12] and [27]). Even though these methods differ in how to partition the data set into blocks, they all considerably reduce the number of comparisons by only comparing records from the same block. Since any of these methods can be incorporated into UDD to reduce the number of record pair comparisons, we do not further consider this issue.

While most previous record matching work is targeted at matching a single type of record, more recent work ([13], [16], and [22]) has addressed the matching of multiple types of records with rich associations between the records. Even though the matching complexity increases rapidly with the number of record types, these works manage to capture the matching dependencies between multiple record types and utilize such dependencies to improve the matching accuracy of each single record type. Unfortunately, however, the dependencies among multiple record types are not available for many domains.

Compared to these previous works, UDD is specifically designed for the Web database scenario where the records to match are of a single type with multiple string fields. These records are heavily query-dependent and are only a partial and biased portion of the entire data, which makes the existing work based on offline learning inappropriate. Moreover, our work focuses on studying and addressing the field weight assignment issue rather than on the similarity measure. In UDD, any similarity measure, or some combination of them, can be easily incorporated.

Our work is also related to the classification problem using only a single class of training examples, i.e., either positive or negative, to find data similar to the given class. To date, most single-class classification work has relied on learning from positive and unlabeled data ([14], [15], and [24]). In [25], multiple classification methods are compared

and it is concluded that one-class SVM and neural network methods are comparable and superior to all the other methods. In particular, one-class SVM distinguishes one class of data from another by drawing the class boundary of the provided data's class in the feature space. However, it requires lots of data to induce the boundary precisely, which makes it liable to overfit or underfit the data and, moreover, it is very vulnerable to noise.

The record matching works most closely related to UDD are Christen's method [8] and PEBL [36]. Using a nearest based approach, Christen first performs a comparison step to generate weight vectors for each pair of records and selects those weight vectors as training examples that, with high likelihood, correspond to either true matches (i.e., pairs with high similarity scores that are used as positive examples) or true nonmatches (i.e., pairs with low similarity scores that are used as negative examples). These training examples are then used in a convergence step to train a classifier (either nearest neighbor or SVM) to label the record pairs not in the training set. Combined, these two steps allow fully automated, unsupervised record pair classification, without the need to know the true match and nonmatch status of the weight vectors produced in the comparison step.

PEBL [36] classifies Web pages in two stages by learning from positive examples and unlabeled data. In the mapping stage, a weak classifier, e.g., a rule-based one, is used to get "strong" negative examples from the unlabeled data, which contain none of the frequent features of the positive examples. In the convergence stage, an internal classifier, e.g., SVM, is first trained by the positive examples and the autoidentified negative examples and is then used to iteratively identify new negative examples until it converges.

The major differences between Christen's method, PEBL, and UDD are:

1. The weights used in Christen's method are static, while in UDD the weights are adjusted dynamically.
2. Both Christen's method and PEBL use only one classifier during the iterations of the convergence stage, while UDD uses two classifiers that cooperate. When there is only one single classifier in the convergence-stage iterations, the classified results from a previous iteration are used by the same classifier as the retraining examples for the next iteration, which makes it unlikely these results will help the classifier obtain a different hypothesis. Using two classifiers cooperatively can help prevent this problem.
3. PEBL is a general framework for the Web page classification problem and it needs a set of positive training examples, while UDD tackles a slightly different classification problem, online duplicate record detection for multiple Web databases. In this scenario, the assumption that *most* records from the same data source are nonduplicates usually holds, i.e., negative examples are assumed without human labeling, which helps UDD overcome the training examples requirement.
4. PEBL assumes that the set of positive training examples is correct; whether and by how much its performance will be affected by false positive examples is not known. Our experiments show that UDD's

TABLE 1
Duplicate Reduction within a Website Using Exact Matching

| Domain | Number of websites | Number of user specified fields | Duplicate ratio | Duplicate pair reduction ratio | Residue duplicate ratio |
|---------------------|--------------------|---------------------------------|-----------------|--------------------------------|-------------------------|
| <i>Book</i> | 10 | 2 | 7.3% | 94% | 0.44% |
| <i>Hotel</i> | 10 | 3 | 3.2% | 76% | 0.77% |
| <i>Movie</i> | 10 | 3 | 6.4% | 96% | 0.26% |
| <i>Music Record</i> | 10 | 3 | 2.1% | 88% | 0.25% |
| <i>Average</i> | 10 | | 4.8% | 89% | 0.55% |

performance is not very sensitive to the false negative cases, i.e., actual duplicates from the same data source.

3 DUPLICATE DETECTION IN UDD

In this section, the duplicate detection problem in the context of Web databases is first defined in Section 3.1, and then, an overview of our solution to this problem, the UDD method, is presented in Section 3.2. Next, the two main classifiers of UDD are described: a weighted component similarity summing classifier in Section 3.3 and an SVM classifier in Section 3.4. The similarity measure used in our experiments is briefly described in Section 3.5.

3.1 Problem Definition

Our focus is on Web databases from the same domain, i.e., Web databases that provide the same type of records in response to user queries. Suppose there are s records in data source A and there are t records in data source B , with each record having a set of fields/attributes. Each of the t records in data source B can potentially be a duplicate of each of the s records in data source A . The goal of duplicate detection is to determine the matching status, i.e., duplicate or nonduplicate, of these $s \times t$ record pairs.

3.1.1 Duplicate Definition

Different users may have different criteria for what constitutes a duplicate even for records within the same domain. For example, in Fig. 1, if the user is only interested in the title and author of a book and does not care about the ISBN information, the records numbered 5 in Fig. 1a and the second record in Fig. 1b are duplicates. Furthermore, the records numbered 5 and 6 in Fig. 1a are also duplicates under this criterion. In contrast, some users may be concerned about the ISBN field besides the title and author fields. For these users, the records numbered 5 and 6 in Fig. 1a and the second record in Fig. 1b are not duplicates. This user preference problem makes supervised duplicate detection methods fail. Since UDD is unsupervised, it does not suffer from this problem.

3.1.2 Assumptions and Observations

In this section, we present the assumptions and observations on which UDD is based. First, we make the following two assumptions:

1. A global schema for the specific type of result records is predefined and each database's individual query

result schema has been matched to the global schema (see, for example, the methods in [19] and [30]).

2. Record extractors, i.e., wrappers, are available for each source to extract the result data from HTML pages and insert them into a relational database according to the global schema (see, for example, the methods in [29], [37], and [38]).

Besides these two assumptions, we also make use of the following two observations:

1. The records from the same data source usually have the same format.
2. Most duplicates from the same data source can be identified and removed using an exact matching method.

Duplicate records exist in the query results of many Web databases, especially when the duplicates are defined based on only some of the fields in a record. Using a straightforward preprocessing step, *exact matching*, can merge those records that are exactly the same in all relevant matching fields.

We investigated 40 Websites for four popular domains on the Web, and as shown in Table 1, found that the simple exact matching step can reduce duplicates by 89 percent, on average. The main reason that exact matching is so effective at reducing duplicates is that the data format for records from the same data source is usually the same for all records. In Table 1, the duplicate ratio is defined as follows:

Definition 1. Suppose there are m records extracted from a data source d . There can be $n = m(m-1)/2$ record pairs generated, which are formed by putting every two records together. Suppose t of the n record pairs are duplicate records. The duplicate ratio of the m records then is t/n .

3.1.3 Problem Formulation

We formulate the duplicate detection problem following the completion of the exact matching step. We represent a pair of records $P_{12} = \{r_1, r_2\}$, where r_1 and r_2 can come from the same or different data sources, as a similarity vector $V_{12} = \langle v_1, v_2, \dots, v_n \rangle$, in which v_i represents the i th field similarity between r_1 and r_2 : $0 \leq v_i \leq 1$. $v_i = 1$ means that the i th fields of r_1 and r_2 are equal and $v_i = 0$ means that the i th fields of r_1 and r_2 are totally different. Note that UDD can employ any similarity function (one or multiple) to calculate the field similarity. The similarity function we use in our experiments is discussed in Section 3.5.

We call a similarity vector formed by a duplicate record pair a *duplicate vector* and a similarity vector formed by a

Input: Potential duplicate vector set P
Non-duplicate vector set N

Output: Duplicate vector set D

C_1 : a classification algorithm with adjustable parameters W that identifies duplicate vector pairs from P

C_2 : a supervised classifier, e.g., SVM

Algorithm:

1. $D = \emptyset$
2. Set the parameters W of C_1 according to N
3. Use C_1 to get a set of duplicate vector pairs d_1 from P
4. Use C_1 to get a set of duplicate vector pairs f from N
5. $P = P - d_1$
6. While $|d_1| \neq 0$
7. $N' = N - f$
8. $D = D + d_1 + f$
9. Train C_2 using D and N'
10. Classify P using C_2 and get a set of newly identified duplicate vector pairs d_2
11. $P = P - d_2$
12. $D = D + d_2$
13. Adjust the parameters W of C_1 according to N' and D
14. Use C_1 to get a new set of duplicate vector pairs d_1 from P
15. Use C_1 to get a new set of duplicate vector pairs f from N
16. $N = N'$
17. Return D

Fig. 2. Duplicate vector identification algorithm.

nonduplicate record pair a *nonduplicate vector*. Initially, two sets of vectors can be built.

1. A nonduplicate vector set N that includes similarity vectors formed by any two different records from the same data source.⁶
2. A potential duplicate vector set P that includes all similarity vectors formed by any two records from different data sources.

Given the nonduplicate vector set N ,⁷ our goal is to try to identify the set of *actual* duplicate vectors D from the potential duplicate vector set P .

3.2 UDD Algorithm Overview

An intuitive solution to this problem is that we can learn a classifier from N and use the learned classifier to classify P . Although there are several works based on learning from only positive (or negative) examples, to our knowledge all works in the literature assume that the positive (or negative) examples are all correct. However, N may contain a small

6. If a large number of result records are returned, a very large number of record pairs will need to be formed. We do not limit the number of nonduplicate vectors in the experiments.

7. Since the simple exact matching method described in Section 3.1.2 is not perfect, i.e., N could contain false negative examples due to spelling errors, etc., the duplicate detection algorithm is designed to consider a flawed negative training set. The experimental results in Section 4.3 show that UDD is still able to perform well even when there are a high percentage of duplicates within a single data source.

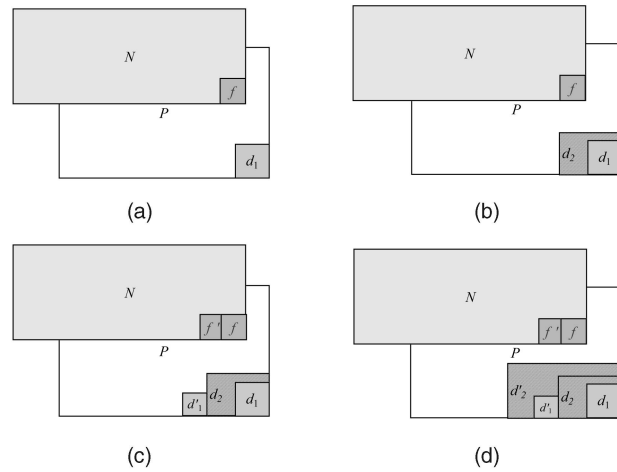


Fig. 3. Duplicate vector identification process. (a) Identify a set of duplicate vectors d_1 from P and f from N using C_1 with parameters learned from N . (b) Identify a set of duplicate vectors d_2 from P with C_2 , which is learned from $N - f$ and D . (c) Identify a new set of duplicate vectors d'_1 from P and f' from N using C_1 with parameters learned from $N - f$ and D . (d) Identify a set of duplicate vectors d'_2 from P using C_2 . The iteration stops when no new duplicate vectors are identified by C_1 .

set of false negative examples. For most general, single-class learning algorithms, such as one-class SVM, these noise examples may have disastrous effects [25].

We propose a method that identifies duplicate vectors in P iteratively, in a way similar to [8] and [36]. However, different from these two works, in which only one classifier is used during the iterations, we employ two classifiers in each iteration that cooperate to identify duplicate vectors from P . Two classifiers are used since we believe that, if there is only one classifier, it is possible that the identified positive instances are not effective enough to retrain the classifier to get a more accurate hypothesis, while two classifiers with different characteristics may discover different sets of positive instances. Thus, the two classifiers can benefit from each other by taking advantage of duplicate vectors identified by the other classifier.

The overall UDD algorithm is presented in Fig. 2. In this algorithm, two classifiers in tandem, C_1 and C_2 , identify the duplicate vector set D iteratively. At the very beginning of the algorithm, the negative example set N is used to set the parameters W of C_1 (line 2). Then, C_1 is used to identify a set of duplicate vectors d_1 from P and a set of duplicate vectors f from N (lines 3 and 4 and Fig. 3a). If there are no duplicates at the very beginning, i.e., $d_1 = \emptyset$, the algorithm will stop at line 6 before any iteration begins. In each iteration, after creating N' by deleting f from N (line 7) and adding d_1 and f into D (line 8), we train C_2 using the updated D and N' (line 9), i.e., using positive and negative examples. Next, we employ the trained C_2 to detect new duplicate vectors d_2 from P (line 10), and then, remove d_2 from P (line 11 and Fig. 3b) and add d_2 to D (line 12). After D is updated, D and N' are used to adjust the parameters W of C_1 (line 13) that were set tentatively according to N before any iteration began. Using C_1 with the new parameters W , we can identify a new set of duplicate vectors from P (d'_1 in Fig. 3c) and a new set of duplicate vectors from N (f' in Fig. 3c), and a new duplicate vector detection iteration begins. The iteration stops when no new duplicates are identified by C_1 (Fig. 3d).

3.3 C_1 —Weighted Component Similarity Summing (WCSS) Classifier

In our algorithm, classifier C_1 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 C_2 to identify new duplicate vectors. Because no duplicate vectors are available initially, classifiers that need class information to train, such as decision tree and Naïve Bayes, 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, such as the record numbered 3 in Fig. 1a and the third record in Fig. 1b. 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. As illustrated in Section 3.1.1, 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. Hence, we define the similarity between records r_1 and r_2 as

$$Sim(r_1, r_2) = \sum_{i=1}^n w_i \cdot v_i, \quad (1)$$

where

$$\sum_{i=1}^n w_i = 1$$

and $w_i \in [0, 1]$ is the weight for the i th similarity component, which represents the importance of the i th field. The similarity $Sim(r_1, r_2)$ between records r_1 and r_2 will be in $[0, 1]$ according to the above definition.

3.3.1 Component Weight Assignment

In the WCSS classifier, we assign a weight to a component to indicate the importance of its corresponding field under the condition that the sum of all component weights is equal to 1. The component weight assignment algorithm is shown in Fig. 4. The intuition for the weight assignment includes:

1. *Duplicate intuition*: The similarity between two duplicate records should be close to 1. For a duplicate vector V_{12} that is formed by a pair of duplicate records r_1 and r_2 , we need to assign large weights to the components with large similarity values and small weights to the components with small similarity values (lines 4-8).
2. *Nonduplicate intuition*: The similarity for two nonduplicate records should be close to 0. Hence, for a nonduplicate vector V_{12} that is formed by a pair of nonduplicate records r_1 and r_2 , we need to assign small weights to the components with large similarity values and large weights to the components with small similarity values (lines 9-14).

According to the duplicate intuition, we design the following weight assignment scheme considering all duplicate vectors in D :

Input: Duplicate vector set D
Non-duplicate vector set N
Weighting scheme co-efficient a

Output: Component Weight W

Algorithm:

1. For $i=1$ to n
2. $p_i=0$
3. $q_i=0$
4. For each vector $V_k=\{v_{k1}, \dots, v_{kn}\}$ in D
5. $p_i = p_i + v_{ki}$
6. $S = \sum_{i=1}^n p_i$
7. For $i=1$ to n
8. $w_{di} = p_i / S$
9. For each vector $V_k=\{v_{k1}, \dots, v_{kn}\}$ in N
10. $q_i = q_i + 1 - v_{ki}$
11. $S = \sum_{i=1}^n q_i$
12. For $i=1$ to n
13. $w_{ni} = p_i / S$
14. $w_i = a \cdot w_{di} + (1 - a)w_{ni}$
15. Return $W=\{w_1, \dots, w_n\}$

Fig. 4. Component weight assignment algorithm.

$$p_i = \sum_{v \in D} v_i \quad (2)$$

and

$$w_{di} = \frac{p_i}{\sum_{j=1}^n p_j} \quad (3)$$

in which p_i is the accumulated i th component similarity value for all duplicate vectors in D (lines 5 and 6) and w_{di} is the normalized weight for the i th component (lines 7 and 8). For each component, if it usually has a large similarity value in the duplicate vectors, p_i will be large according to (2) and, in turn, a large weight will be assigned for the i th component according to (3). On the other hand, the component will be assigned a small weight if it usually has a small similarity value in the duplicate vectors.

According to the nonduplicate intuition, we use the following weight assignment scheme considering all nonduplicate vectors in N :

$$q_i = \sum_{v \in N} (1 - v_i) \quad (4)$$

and

$$w_{ni} = \frac{q_i}{\sum_{j=1}^n q_j} \quad (5)$$

in which q_i is the accumulated i th component dissimilarity value for all nonduplicate vectors in N (lines 10 and 11) and w_{ni} is the normalized weight for the i th component (lines 12 and 13). In a similarity vector $V = \langle v_1, v_2, \dots, v_n \rangle$, the i th component dissimilarity refers to $1 - v_i$. For each component, if it usually has a large similarity value in the

nonduplicate vectors, it will have a small accumulated dissimilarity according to (4) and will, in turn, be assigned a small weight according to (5). On the other hand, it will be assigned a large weight if it usually has a small similarity value in the nonduplicate vectors.

In general, the two intuitions will give rise to different weighting schemes, which need to be combined to generate a more reasonable weighting scheme. In our experiment, we give each scheme a weight to show its importance (line 14):

$$w_i = a \cdot w_{di} + (1 - a)w_{ni} \quad (6)$$

in which $a \in [0, 1]$ denotes the importance of duplicate vectors versus nonduplicate vectors. At the start of our algorithm in Fig. 2, there is no duplicate vector available. Hence, a is assigned to be 0. As more duplicate vectors are discovered, we increase the value of a . We initially set a to be 0.5 at the 2nd iteration to indicate that D and N are equally important and incrementally add 0.1 for each of the subsequent iterations.⁸

3.3.2 Duplicate Identification

After we assign a weight for each component, the duplicate vector detection is rather intuitive. Two records r_1 and r_2 are duplicates if $\text{Sim}(r_1, r_2) \geq T_{\text{sim}}$, i.e., if their similarity value is equal to or greater than a similarity threshold. In general, the similarity threshold T_{sim} should be close to 1 to ensure that the identified duplicates are correct. Increasing the value of T_{sim} will reduce the number of duplicate vectors identified by C_1 while, at the same time, the identified duplicates will be more precise. The influence of T_{sim} on the performance of our algorithm will be illustrated in Section 4.3.3.

3.4 C_2 —Support Vector Machine Classifier

After detecting a few duplicate vectors whose similarity scores are bigger than the threshold using the WCSS classifier, we have positive examples, the identified duplicate vectors in D , and negative examples, namely, the remaining nonduplicate vectors in N' . Hence, we can train another classifier C_2 and use this trained classifier to identify new duplicate vectors from the remaining potential duplicate vectors in P and the nonduplicate vectors in N' .

A classifier suitable for the task should have the following characteristics. First, it should not be sensitive to the relative size of the positive and negative examples because the size of the negative examples is usually much bigger than the size of the positive examples. This is especially the case at the beginning of the duplicate vector detection iterations when a limited number of duplicates are detected. Another requirement is that the classifier should work well given limited training examples. Because our algorithm identifies duplicate vectors in an iterative way, any incorrect identification due to noise during the first several iterations, when the number of positive examples is limited, will greatly affect the final result.

According to [33], Support Vector Machine (SVM), which is known to be insensitive to the number of training

examples, satisfies all the desired requirements and is selected for use in UDD. Because our algorithm will be used for online duplicate detection, we use a linear kernel, which is the fastest, as the kernel function in our experiments.

3.5 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 (S_a, S_b) , a similarity function calculates the similarity score between S_a and S_b , 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 in UDD (e.g., [6] and [18]). 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 [31].

Specifically, in our experiments, a transformation-based similarity calculation method is adapted from [31] in which, given two strings $S_a = \{t_{a1}, t_{a2}, \dots, t_{am}\}$ and $S_b = \{t_{b1}, t_{b2}, \dots, t_{bn}\}$ containing a set of tokens, a string transformation from S_a to S_b is a sequence of operations that transforms the tokens of S_a to tokens of S_b . Among others, the string transformations considered include *Initial* (one token is equal to the first character of the other), *Substring* (one token is a substring of the other), and *Abbreviation* (characters in one token are a subset of the characters in the other token). After the k transformation steps are identified, the similarity score between S_a and S_b is calculated as the cosine similarity between the token vectors S_a and S_b with each token associated with a TF-IDF weight, where TF means the frequency of a token in a string and IDF the inverse of the number of strings that contain the token in the field. The TF value of a token is usually 1 and IDF will reduce the weight for tokens, such as the tokens in the user query that appear in multiple strings in a field.

4 EXPERIMENTS

This section describes the experiments used to validate the UDD method. First, we introduce the data sets used in Section 4.1. Then, Section 4.2 describes the evaluation metric. Finally, the experimental results are presented in Section 4.3, which includes the influence of different parameters of our algorithm on the final results.

4.1 Data Sets

We tested UDD on five data sets. To compare UDD with existing approaches, we ran our experiments on *Cora*, which is a collection of research paper citations provided by [26] and whose subsets have been used in [5], [12], and [13]. Each citation of a paper originally was a text string and was manually segmented into multiple fields such as *Author*, *Title*, *Year*, etc. The paper coreferences, i.e., duplicates, which are citations that refer to the same paper, were manually labeled. Unfortunately, *Cora* is a noisy data set in which there are some fields that are not correctly segmented and there is some erroneous coreference labeling. In our experiments, we separate the *Cora* data set into different subsets to make sure that our assumption that the records

8. To show the impact of nonduplicate pairs, the maximum value of a should be larger than 0.5 and smaller than 1, but close to 1. Since most iterations stop at the 2nd or 3rd iteration, we add 0.1 in each iteration so that a can approach 1. The best result is obtained when a is 0.8.

TABLE 2
Properties of the Data Sets

| Dataset | Number of entities | Number of records | Number of fields | Number of sources | Max number of duplicates for an entity* |
|-------------------|--------------------|-------------------|------------------|-------------------|-----------------------------------------|
| <i>Cora</i> | 194 | 1878 | 16 | 119* | 119 |
| <i>Book-full</i> | 840 | 4356 | 4 | 20 | 11 |
| <i>Book-titau</i> | 540 | 2659 | 2 | 20 | 14 |
| <i>Hotel</i> | 780 | 3896 | 3 | 10 | 10 |
| <i>Movie</i> | 156 | 587 | 9 | 10 | 9 |

* The Max number of duplicates for an entity in the Cora dataset is 119. Hence, we divided the Cora dataset into 119 parts so that there is no duplicate in any part.

from each data source are distinct holds. We assign the records into n subsets, where n is the largest number of duplicates that a paper has in Cora. Each duplicate is randomly assigned to a subset under the restriction that a paper's duplicates cannot be assigned to the same subset.

To show the applicability of our algorithm for detecting duplicate query results collected from Web databases, we also collected some real Web data from Web databases in three domains: Book, Hotel, and Movie. For the Book domain, we collected records with Title, Author, Publisher, and ISBN fields. In our experiments, we run the UDD algorithm on the full data set with all four fields (referred to as *Book-full*) as well as on the subdata set with only fields Title and Author (referred to as *Book-titau*). For the Hotel domain, we collected records with Name, Address, and Star fields from 10 hotel-booking Web sites. For the Movie domain, we collected records with nine fields such as Title, Actor, Director, etc., from 10 movie-selling Web sites. Some of the databases do not provide values for some fields in their query results. In such cases, we leave these fields blank considering them to have a *null* value. When measuring the similarity of a record that has a null value field to another record, we set the component value for the null value field to be the average of all other nonnull fields' component values in the similarity vector.

The properties of these data sets are summarized in Table 2. The results shown in later tables are the performance of the duplicate detection algorithms averaged over all the queries for each domain.

In our experiments, UDD is implemented in C++ and was run on a PC with an Intel 3.0 GHz CPU and 1 GB RAM. The operating system is Mandriva Linux 2005 Limited Edition and the compiler is GCC 3.4.3. It was observed that UDD could finish the record matching task within 1 second for each query, on average, which is quite acceptable in the Web database record matching scenario.

4.2 Evaluation Metric

As in many other duplicate detection approaches, we report the overall performance using *recall* and *precision*, which are defined as follows:

$$\text{precision} = \frac{\# \text{of Correctly Identified Duplicate Pairs}}{\# \text{of All Identified Duplicate Pairs}}$$

$$\text{recall} = \frac{\# \text{of Correctly Identified Duplicate Pairs}}{\# \text{of True Duplicate Pairs}},$$

where the ground truth, i.e., *#of True Duplicate Pairs*, for the Cora data set is given in [26] and that for the other data sets is manually identified.

However, as indicated in [10], due to the usually imbalanced distribution of matches and nonmatches in the weight vector set, these commonly used accuracy measures are not very suitable for assessing the quality of record matching. The large number of nonmatches usually dominates the accuracy measure and yields results that are too optimistic. Thus, we also use the *F-measure*, which is the harmonic mean of precision and recall, to evaluate the classification quality [2]:

$$F - \text{measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}.$$

4.3 Experimental Results

4.3.1 Cora Data Set

We ran UDD on the Cora data set and its three subsets individually when the similarity threshold $T_{sim} = 0.85$. Although Cora is a noisy data set, our algorithm still performs well over it. UDD has a precision of 0.896, recall of 0.950, and F-measure of 0.923 over the Cora data set. We compared our results with other works that use all or part of the Cora data set. Bilenko and Mooney [5], in which a subset of the Cora data set is used, report an F-measure of 0.867. Cohen and Richman [12] report 0.99/0.925 for precision/recall using a subset of the Cora data set. Culotta and McCallum [13] report an F-measure of 0.908 using the full Cora data set. From this comparison, it can be seen that the performance of UDD is comparable to these methods, all of which require training examples.

4.3.2 Web Database Data Sets

Table 3 shows the precision, recall, F-measure value, and actual execution time of UDD on the Web database data sets when the similarity threshold $T_{sim} = 0.85$. It can be seen that UDD can efficiently identify duplicates among records from multiple data sources with good precision and recall, on average.

4.3.3 Performance Evaluation

Effect of the threshold T_{sim} . In Section 3.3.2, it is assumed that two records r_1 and r_2 are duplicates if their similarity $Sim(r_1, r_2)$ is larger than or equal to a similarity threshold

TABLE 3
Performance of UDD on the Web Database Data Sets

| | Precision | Recall | F-measure | Avg. Execution Time (sec) |
|-------------------|-----------|--------|-----------|---------------------------|
| <i>Book-full</i> | 0.954 | 0.925 | 0.939 | 0.85 |
| <i>Book-titau</i> | 0.947 | 0.952 | 0.950 | 0.36 |
| <i>Hotel</i> | 0.961 | 0.952 | 0.955 | 0.74 |
| <i>Movie</i> | 0.932 | 0.928 | 0.930 | 0.21 |

T_{sim} . Table 4 shows UDD's performance when using five different similarity thresholds (T_{sim} : 0.75, 0.80, 0.85, 0.90, and 0.95) on the four Web database data sets. The iteration row in this table indicates the number of iterations required for UDD to stop.

It can be seen that the duplicate vector detection iterations stop very quickly. All of them stop by the fifth iteration. On the one hand, the smaller T_{sim} is, the more iterations are required and the higher the recall. This is because the WCSS classifier with smaller T_{sim} identifies more duplicates and most of them are correct duplicates. Hence, with more correct positive examples, the SVM classifier can also identify more duplicates, which, in turn, results in a higher recall. On the other hand, the smaller T_{sim} is, the lower the precision. This is because the WCSS classifier with smaller T_{sim} is more likely to identify incorrect duplicates, which may incorrectly guide the SVM classifier to identify new incorrect duplicates. In our experiments, the highest F-measures were achieved when $T_{sim} = 0.85$ over all the four data sets.

We also observe from Table 4 that the iterations stop more quickly when the threshold T_{sim} is high. When T_{sim} is high, vectors are required to have more large similarity values on their fields in order to be identified as duplicates in the early iterations. Hence, vectors with only a certain number of fields having large similarity values and other fields having small similarity values are likely to stay in the negative example set N . Recall that, according to the nonduplicate intuition, when setting the component weights in the WCSS classifier, fields with more large similarity values in N gain smaller weights and fields with more small similarity values in N gain larger weights. The vectors that stay in N would make the fields with small similarity values on all vectors in N relatively even smaller compared with other fields. Thus, they gain larger weights. In turn, the vectors staying in N are even more unlikely to be identified as duplicates in the next iteration because of the larger weights on their fields with small similarity values. Consequently, a high T_{sim} value makes it difficult for the WCSS classifier to find new positive instances after the first two iterations.

Influence of duplicate records from the same data source. Recall that, in Section 3, we assumed that records from the same data source are nonduplicates so that we can put pairs of them into the negative example set N . However, we also pointed out that, in reality, some of these record pairs could be actual duplicates. We call the actual duplicate vectors in N *false nonduplicate vectors*. The number of false

TABLE 4
Performance of UDD with Different T_{sim} on the Web Database Data Sets

| | T_{sim} | 0.75 | 0.8 | 0.85 | 0.9 | 0.95 |
|-------------------|-----------|------|------|------|------|------|
| <i>Book-full</i> | iteration | 4 | 4 | 2 | 2 | 2 |
| | precision | .767 | .80 | .954 | .955 | .959 |
| | recall | .964 | .935 | .925 | .908 | .874 |
| | F-measure | .854 | .862 | .939 | .931 | .914 |
| <i>Book-titau</i> | iteration | 4 | 3 | 2 | 2 | 2 |
| | precision | .755 | .806 | .954 | .961 | .969 |
| | recall | .951 | .947 | .948 | .908 | .854 |
| | F-measure | .842 | .871 | .951 | .934 | .908 |
| <i>Hotel</i> | iteration | 5 | 4 | 3 | 2 | 2 |
| | precision | .732 | .865 | .956 | .962 | .963 |
| | recall | .987 | .965 | .950 | .902 | .807 |
| | F-measure | .841 | .912 | .953 | .931 | .878 |
| <i>Movie</i> | iteration | 5 | 5 | 4 | 3 | 2 |
| | precision | .717 | .817 | .943 | .984 | .986 |
| | recall | .965 | .942 | .921 | .85 | .80 |
| | F-measure | .823 | .875 | .932 | .912 | .882 |

nonduplicate vectors increases as the duplicate ratio, defined in Definition 1 in Section 3.1.2, increases.

The false nonduplicate vectors will affect the two classifiers in our algorithm in the following ways:

1. While setting the component weights W in WCSS according to the nonduplicate intuition, components that usually have large similarity values in N will be assigned small weights. Note that the false nonduplicate vectors are actually duplicate vectors with large similarity values for their components. As a result, an inappropriate set of weights could be set for WCSS.
2. Since the SVM classifier learns by creating a hyperplane between positive and negative examples, the false nonduplicate vectors will incorrectly add positive examples in the negative space. As a result, the hyperplane could be moved toward the positive space and the number of identified duplicate vectors will be much smaller.

Deleting the false nonduplicate vectors from N in the algorithm in Fig. 2 (line 7) can alleviate this effect. Table 5 shows the performance of UDD under different duplicate ratios: 0, 2, 5, and 10 percent. As expected, the performance decreases as the duplicate ratio increases. However, if the duplicate ratio is small, the performance is not greatly affected, e.g., when the percentage is 2 percent. The F-measure is still larger than 0.88 even when the duplicate ratio is 10 percent, which is actually a quite high duplicate ratio, as illustrated by Example 1.

Example 1. Suppose there is a data set with 100 records that are extracted from the same data source in which every

TABLE 5
Performance of UDD on the Book-Full Data Set under Different Duplicate Ratios ($T_{sim} = 0.85$)

| Duplicate ratio | Precision | Recall | F-measure |
|-----------------|-----------|--------|-----------|
| 0% | 0.954 | 0.925 | 0.939 |
| 2% | 0.945 | 0.914 | 0.929 |
| 5% | 0.910 | 0.898 | 0.904 |
| 10% | 0.880 | 0.885 | 0.882 |

10 records represent the same entity, i.e., the data set contains only 10 distinct records. There will be $100 \times 99/2 = 4,950$ record pairs, i.e., nonduplicate vectors, generated in total containing $(10 \times 9/2) \times 10 = 450$ false nonduplicate vectors. That is, the duplicate ratio of this data set is $450/4950 = 9.09$ percent.

Effect of the number of iterations. Fig. 5 shows the performance of UDD at the end of each duplicate detection iteration over the four Web database data sets when $T_{sim} = 0.85$. It can be seen that the iteration stops quickly for all data sets and takes at most four iterations.

4.3.4 Comparison with Other Classification Methods

Table 6 shows the precision, recall, and F-measure for UDD compared with four other machine learning approaches, SVM, PEBL, OSVM, and Christen's method, on the Book-full data set. SVM refers to a normal SVM based on 1,000 duplicate and 1,000 nonduplicate vectors being provided as training examples, which are randomly selected from the ground truth. OSVM [25], refers to one-class SVM that is provided with the nonduplicate vectors in N as examples to find vectors in P , say N_p , that are similar to the provided examples. Since N_p is the set of vectors that are similar to N , i.e., nonduplicates, $P - N_p$ is the actual set of duplicate vectors identified by OSVM. In [36], PEBL learns from positive examples and unlabeled data, with one weak classifier before the iterations and one classifier inside the iterations. In our experiments, we implement it to learn from the nonduplicate examples, N , and the unlabeled data, P , by using the WCSS classifier as the weak classifier to identify "strong" positive examples and using an SVM classifier as the second classifier. For Christen's method, we use the classifier that has the best

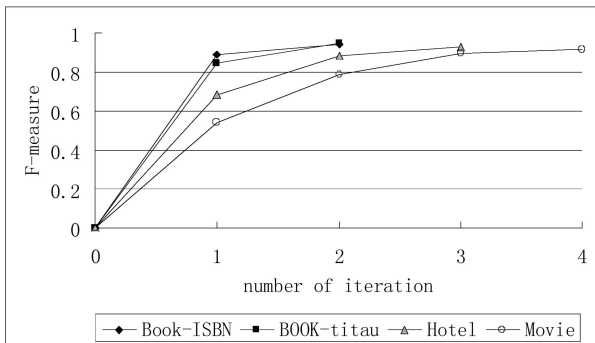


Fig. 5: Performance of UDD at the end of each duplicate detection iteration over the four data sets when $T_{sim} = 0.85$.

TABLE 6
Performance Comparison between UDD and Other Learning Methods on the Book-Full Data Set ($T_{sim} = 0.85$)

| | Precision | Recall | F-measure | Avg. Execution Time (sec) |
|----------|-----------|--------|-----------|---------------------------|
| UDD | 0.924 | 0.915 | 0.919 | 0.85 |
| SVM | 0.926 | 0.933 | 0.929 | 0.36 |
| OSVM | 0.580 | 0.460 | 0.513 | 0.42 |
| PEBL | 0.902 | 0.803 | 0.851 | 1.42 |
| Christen | 0.886 | 0.867 | 0.876 | 1.64 |

performance in [8], i.e., a two-step iterative SVM-based classifier with increment percentage $ip = 50$ percent and total training percentage $tp = 100$ percent. For efficiency reasons, all SVMs use the linear kernel function in our experiments.

It can be seen that the performance of UDD is close to the performance of SVM on the Book-full data set because UDD first produces high-quality positive and negative instances using one classifier, and then, uses an SVM classifier to do classification. However, UDD has the advantage that it does not require any prelabeled training examples, which relieves the burden on users having to provide such examples and makes UDD applicable for online record matching in the Web database scenario.

OSVM totally fails at this task even though most state-of-the-art work shows the effectiveness of single-class SVM when it is provided with a large number of positive examples to identify new positive data. However, in the current task scenario, there are only negative examples, i.e., nonduplicate vectors in N , for OSVM to learn from. Moreover, N may actually have some false nonduplicate vectors and, since OSVM is vulnerable to noise, it fails more easily.

UDD outperforms both PEBL and Christen's method on both precision and recall, and for PEBL especially on the recall. While both PEBL and UDD employ two classifiers, the two classifiers in UDD alternately cooperate inside the iterations while the weak classifier in PEBL only works before the iterations. Thus, UDD outperforms PEBL since in UDD either classifier can identify instances that cannot be identified by the other classifier, which reduces the possibility of being biased by the false positive/negative examples. In both PEBL and Christen's method, there is only one classifier in the iterations. When there is only one classifier in the iterations, the identified results in a previous iteration are used by the same classifier as the new training examples for the next iteration, which makes the classifier more vulnerable to false examples.

It can also be seen that UDD is slower than SVM and OSVM because UDD needs two iterations to identify the duplicates. However, UDD is faster than PEBL and Christen's method, which require more iterations than UDD to identify all duplicates.

5 CONCLUSIONS

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. Experimental results show that our approach is comparable to previous work that requires training examples for identifying duplicates from the query results of multiple Web databases.

ACKNOWLEDGMENTS

This research was supported by the Research Grants Council of Hong Kong under grant HKUST6172/04E.

REFERENCES

- [1] 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.
- [2] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, 1999.
- [3] R. Baxter, P. Christen, and T. Churches, "A Comparison of Fast Blocking Methods for Record Linkage," *Proc. KDD Workshop Data Cleaning, Record Linkage, and Object Consolidation*, pp. 25-27, 2003.
- [4] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S.E. Whang, and J. Widom, "Swoosh: A Generic Approach to Entity Resolution," *The VLDB J.*, vol. 18, no. 1, pp. 255-276, 2009.
- [5] M. Bilenko and R.J. Mooney, "Adaptive Duplicate Detection Using Learnable String Similarity Measures," *Proc. ACM SIGKDD*, pp. 39-48, 2003.
- [6] 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.
- [7] S. Chaudhuri, V. Ganti, and R. Motwani, "Robust Identification of Fuzzy Duplicates," *Proc. 21st IEEE Int'l Conf. Data Eng.*, pp. 865-876, 2005.
- [8] P. Christen, "Automatic Record Linkage Using Seeded Nearest Neighbour and Support Vector Machine Classification," *Proc. ACM SIGKDD*, pp. 151-159, 2008.
- [9] 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.
- [10] 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.
- [11] W.W. Cohen, H. Kautz, and D. McAllester, "Hardening Soft Information Sources," *Proc. ACM SIGKDD*, pp. 255-259, 2000.
- [12] 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.
- [13] A. Culotta and A. McCallum, "A Conditional Model of Deduplication for Multi-Type Relational Data," Technical Report IR-443, Dept. of Computer Science, Univ. of Massachusetts Amherst, 2005.
- [14] F. DeComite, F. Denis, and R. Gilleron, "Positive and Unlabeled Examples Help Learning," *Proc. 11th Int'l Conf. Algorithmic Learning Theory*, pp. 219-230, 1999.
- [15] F. Denis, "PAC Learning from Positive Statistical Queries," *Proc. 10th Int'l Conf. Algorithmic Learning Theory*, pp. 112-126, 1998.
- [16] X. Dong, A. Halevy, and J. Madhavan, "Reference Reconciliation in Complex Information Spaces," *Proc. ACM SIGMOD*, pp. 85-96, 2005.
- [17] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios, "Duplicate Record Detection: A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 1, pp. 1-16, Jan. 2007.
- [18] 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.
- [19] B. He and K.C.-C. Chang, "Automatic Complex Schema Matching Across Web Query Interfaces: A Correlation Mining Approach," *ACM Trans. Database Systems*, vol. 31, no. 1, pp. 346-396, 2006.
- [20] M.A. Hernandez and S.J. Stolfo, "The Merge/Purge Problem for Large Databases," *ACM SIGMOD Record*, vol. 24, no. 2, pp. 127-138, 1995.
- [21] M.A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida," *J. Am. Statistical Assoc.*, vol. 89, no. 406, pp. 414-420, 1989.
- [22] D.V. Kalashnikov, S. Mehrotra, and Z. Chen, "Exploiting Relationships for Domain-Independent Data Cleaning," *Proc. SIAM Int'l Conf. Data Mining*, pp. 262-273, 2005.
- [23] N. Koudas, S. Sarawagi, and D. Srivastava, "Record Linkage: Similarity Measures and Algorithms (Tutorial)," *Proc. ACM SIGMOD*, pp. 802-803, 2006.
- [24] F. Letouzey, F. Denis, and R. Gilleron, "Learning from Positive and Unlabeled Examples," *Proc. 11th Int'l Conf. Algorithmic Learning Theory*, pp. 71-85, 2000.
- [25] L.M. Manevitz and M. Yousef, "One-Class SVMs for Document Classification," *J. Machine Learning Research*, vol. 2, pp. 139-154, 2001.
- [26] A. McCallum, "Cora Citation Matching," <http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz>, 2004.
- [27] A. McCallum, K. Nigam, and L.H. Ungar, "Efficient Clustering of High-Dimensional Datasets with Application to Reference Matching," *Proc. ACM SIGKDD*, pp. 169-178, 2000.
- [28] S. Sarawagi and A. Bhamidipaty, "Interactive Deduplication Using Active Learning," *Proc. ACM SIGKDD*, pp. 269-278, 2002.
- [29] K. Simon and G. Lausen, "ViPER: Augmenting Automatic Information Extraction with Visual Perceptions," *Proc. 14th ACM Int'l Conf. Information and Knowledge Management*, pp. 381-388, 2005.
- [30] W. Su, J. Wang, and F.H. Lochovsky, "Holistic Schema Matching for Web Query Interfaces," *Proc. 10th Int'l. Conf. Extending Database Technology*, pp. 77-94, 2006.
- [31] S. Tejada, C.A. Knoblock, and S. Minton, "Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification," *Proc. ACM SIGKDD*, pp. 350-359, 2002.
- [32] Y. Thibaudau, "The Discrimination Power of Dependency Structures in Record Linkage," *Survey Methodology*, vol. 19, pp. 31-38, 1993.
- [33] V. Vapnik, *The Nature of Statistical Learning Theory*, second ed. Springer, 2000.
- [34] V.S. Verykios, G.V. Moustakides, and M.G. Elfekey, "A Bayesian Decision Model for Cost Optimal Record Matching," *The VLDB J.*, vol. 12, no. 1, pp. 28-40, 2003.
- [35] W.E. Winkler, "Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage," *Proc. Section Survey Research Methods*, pp. 667-671, 1988.
- [36] H. Yu, J. Han, and C.C. Chang, "PEBL: Web Page Classification without Negative Examples," *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 1, pp. 70-81, Jan. 2004.
- [37] Y. Zhai and B. Liu, "Structured Data Extraction from the Web Based on Partial Tree Alignment," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 12, pp. 1614-1628, Dec. 2006.
- [38] H. Zhao, W. Meng, A. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines," *Proc. 14th World Wide Web Conf.*, pp. 66-75, 2005.



Weifeng Su received the BSc degree in computer science from the Petroleum University of China in 1995, the MSc degree from Xiamen University of China in 2002, and the PhD degree from The Hong Kong University of Science and Technology in 2007. He is an assistant professor in the Computer Science and Technology Program in BNU-HKBU-UIC and an associate researcher of Shenzhen Key Laboratory of Intelligent Media and Speech, PKU-HKUST Shenzhen Hong Kong Institution. His research interests include database, deep Web, data mining, machine learning, and natural language processing.



Jiying Wang received the BS degree in computer science from Peking University, China, in 1999, and the PhD degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 2004. She is currently lecturing at the Department of Computer Science, City University of Hong Kong. Her research interests include Web search, Web information extraction, the deep Web, data integration, and information retrieval.



Frederick H. Lochovsky received the BASc degree in engineering science and the MSc and PhD degrees in computer science from the University of Toronto. He is currently a professor in the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. He was previously a professor in the Department of Computer Science at the University of Toronto. His research interests include hidden Web search and data integration, information systems design, database design, and knowledge representation. He is a member of the IEEE Computer Society.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.