A KNOWLEDGE BASED ONLINE RECORD MATCHING OVER QUERY RESULTS FROM MULTIPLE WEB DATABASE

M.Ann Michle, K.C. Abhilash Sam Paulstín
Nanjil Catholic College of Arts and Science, Kaliyakkavilai.

**Objective**-The goal of this thesis is to develop a mechanism that can identify duplicate record pairs in the query results that refer to the same real world entity using unsupervised algorithm. The main challenging of this is to reduce the duplicates from different Uniform Resource Locator. To reduce this duplicate record we are using the current system.

**Abstract**-In the Web database scenario, the records to match are highly query dependent, since they can only be obtained through online queries. Consequently, hand-coding or offline approaches are not appropriate for two reasons. First, the full data set is available beforehand, and therefore, good representative data for training are hard to obtain. Second, and most importantly, even if good 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 as 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 of other fields or even is zero. The two classifiers can benefit from each other by talking advantage of duplicate vectors identified by the other classifier.UDD has advantage that it does not require any pre labelled training examples ,which relieves the burden on user having to providing such examples and makes UDD applicable for online record matching in the Web database.

**I. INTRODUCTION**

There are now many searchable databases on the Web. These databases are accessed through queries formulated on their query interfaces only which are usually query forms. The query results from these databases are dynamically generated Web pages is estimated around 500 times the number of static Web pages on the surface web. In many domains, users are interested in obtaining information from multiple sources. Thus, they have to access different Web databases individually via their query interfaces. For large-scale data integration over the Deep Web, it is not practical to manually model and integrate these Web databases. I aim to provide a uniform query interface that allows users to have uniform access to multiple sources. Users can submit their queries to the uniform query interface and be responded with a set of combined results from multiple sources automatically. Schema matching across query interfaces is a critical step in Web data integration, which finds attribute correspondences between the two schemes as input and produces a set of attribute correspondences between the two schemes. The problem of scheme matching has been extensively studied. Some of these methods make use of information about schemes, including structures, linguistic features, data types, value ranges, etc to match attributes between schemes.

Match results from individual matchers are not accurate and certain, because they rely on individual aspects of information about schemas only, which are not sufficient for finding attribute correspondences between schemas. Individual matchers however can generate some degree of belief on the validity of possible attribute correspondences.
II. IMPLEMENTATION DETAILS

To implement the problem of record matching in the web scenario, we present an unsupervised, online record matching, 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” non duplicate 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 non duplicate set, use two cooperating classifiers, a Weighted Component Similarity Summing(WCSS) classifier and an Support Vector Machine(SVM) classifier, to iteratively identify duplicates in the query results from multiple web databases.

2.1. Similarity Vector Calculation

Similarity vector calculation holds comparisons of two records. Inputs to this process are the potential duplicate dataset and non-duplicate dataset.UDD algorithm calculates the similarity of record pair in both dataset. The output of this process serves as input to weighted Component Similarity Classifier (WCSS) and SVM classifier.

Similarity vector calculation involves comparing each field in a record to the corresponding field in another record. We represent a pair of records s12={r1,r2}, where r1 and r2 can come from the same or different data sources, as a similarity vector V1<v1,v2,v3....vn> in which vi represents the ith field similarity between r1 and r20<=vi<=1,Vi=1 means that the ith fields of r1 and r2 are equal and vi=0 means that the ith fields of r1 and r2 are totally different.UDD can employ any similarity function to calculate the fields similarity.

Given a pair of string (Sa ,Sb), a similarity function calculates the similarity score between Sa and Sb , which must be between 0 and 1. In my experiments, a transformation based similarity calculation method is adapted in which, given two string Sa=(ta1,ta2,.....tam} and Sb={tb1,tb2,tb3,......tbm} containing a set of tokens, A string transformation from Sa to Sb is a sequence of operations that perform the tokens of Sa to tokens of Sb.

<table>
<thead>
<tr>
<th>Restaurant Name</th>
<th>Address</th>
<th>City</th>
<th>Phone</th>
<th>Cuisine</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Campanile spice restaurant</td>
<td>654.s.La.breaave</td>
<td>Los angles</td>
<td>213-938-1447</td>
<td>California</td>
<td>FODORS</td>
</tr>
<tr>
<td>Campanile spices</td>
<td>624 s la breave</td>
<td>Los angles</td>
<td>213-938-1447</td>
<td>american</td>
<td>ZAGAT</td>
</tr>
</tbody>
</table>

0.5

2.2. Text Transformation

Some of the text transformation types that can be used to compare two words (token). That are equality tests, stemming converts, substring computation and drop. The process of transformation can be explained with the example given below.
Both the strings are converted to lower case and any special characters are removed.

Split each of the strings into tokens using the white space as delimiter.

Each of the token from FODORS records is compared with all the token in ZAGAT’s to determine if text transformation exists between tokens.

Once the transformations are determined between the tokens, the transformation count is increased by 1 and the tokens are deleted.

In our example equality transformation exists for the first token of FODORS record and first token of ZAGAT’s record as they are same. Stemming is performed on the second token and drop is performed on the third token.

The total number of transformation is calculated. In our example that would be total of 3 transformations.

2.3 Computing Attribute Similarity Scores

Once the number of transformation is obtained, we can apply this count for calculating the attribute similarity scores. Here we use the cosine measure which is commonly used in information retrieval engines. Similarity score for a pair of attribute value is computed using attribute similarity formula.

Where A and B are two attribute values, count of transformations are $w_{ia}$ and $w_{ib}$. If the term exists frequency is 1 otherwise 0. Outputs of attribute similarity scores are sent to the next function for calculate the total object similarity score as a weighted sum of the attribute similarity scores.

2.4 Weighted Component Similarity Summing (WCSS) Classifier

WCSS is algorithm in identifying duplicates. Inputs are similarity scores from potential duplicate and non duplicate sets. We want to develop an unsupervised method for this classifier. The output from this classifier is a duplicate dataset identified from the potential duplicates and non duplicate sets. Once the similarity vector are found WCSS and SVM classifiers iterate until no further duplicates are to be found.

2.5 Attribute Weight Assignment

In the WCSS classifier, we assign weight to an attribute to indicate its importance. The weight of a attribute are given in such a way that a sum of all attribute weight is equal to 1. WCSS classifier employs duplicate and non duplicate intuitions for assigning weights. Inputs to this function are the similarity vector of non duplicate records and duplicate record.

Duplicate Intuition:
For duplicate records the similarity between them should be close to 1.

Equation for weight calculation for all fields using duplicate vector

$$W_{di} = \frac{P_i}{\sum_{j=1}^{m} P_j}$$

//m is the number of fields in dataset
Pi = Σn_{k=1}^{n} V_{ki} \quad /n \text{ is the number of records in duplicate vector}

Where \( W_{di} \) = Normalized weight for \( i^{th} \) attribute

\( Pi = \text{Accumulated } i^{th} \text{ attribute similarity value for all duplicate vectors} \)

**Non-Duplicate Intuition:**
In non duplicate records the similarity between them should be close to 0.

\( W_{ni} = \frac{q_{i}}{\sum_{j=1}^{m} q_{j}} \quad /m \text{ is the number of fields in dataset} \)

\( q_{i} = \sum_{k=1}^{n} (1-V_{ki}) \quad /n \text{ is the number of records in duplicate vector} \)

Where \( W_{ni} \) = Normalized weight for \( i^{th} \) attribute

\( q_{i} = \text{Accumulated } i^{th} \text{ attribute similarity value for all non duplicate vectors} \)

**Sample dataset**

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Address</th>
<th>City</th>
<th>Phone</th>
<th>Cuisine</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anjappar</td>
<td>First street, Anna Nager</td>
<td>Chennai</td>
<td>26553321</td>
<td>Indian</td>
<td>ABC</td>
</tr>
<tr>
<td>Anjappar</td>
<td>First street, Anna Nager</td>
<td>Chennai</td>
<td>26553321</td>
<td>Indian</td>
<td>ACD</td>
</tr>
<tr>
<td>KFC</td>
<td>Sky Walky</td>
<td>Chennai</td>
<td>2323232</td>
<td>Chinese</td>
<td>ABC</td>
</tr>
<tr>
<td>KFC</td>
<td>Sky Walky</td>
<td>Chennai</td>
<td>2454544</td>
<td>Chinese</td>
<td>ACD</td>
</tr>
</tbody>
</table>

Calculate weight for fields “restaurant name” and “city” in non duplicate vector set

\( q_{\text{name}} = \sum_{k=1}^{n} (1-V_{ki}) \)

\( = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)=6 \)

\( q_{\text{address}} = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.145)+(1-0.0)=5.85 \)

\( q_{\text{city}} = (1-1)+(1-1)+(1-1)+(1-1)+(1-1)=0 \)

\( q_{\text{phone}} = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)=6 \)

\( q_{\text{cuisine}} = (1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)+(1-0.0)=6 \)

\( \sum_{j=1}^{m} q_{j} = 6+5.85+0+6+6=23.85 \)

\( \text{Weight}_{\text{name}} = 6/23.85 = .25 \)

\( \text{Weight}_{\text{city}} = 0/23.85 = 0 \)

Weight of city is 0 because it has high similarity vectors for all the records and restaurant name has low similarity vector.

**Duplicate identification**

Once we get the weights of each field and the similarity vectors of non duplicate and potential duplicate data sets. The duplicate detection can be done by calculation the similarity between the records. Hence we define the similarity between the records as

\( \text{Similarity}(r_{1},r_{2}) = \sum_{i=1}^{n} W_{i} * V_{i} \)

Equation: WCSS similarity for records r1 and r2
Where \( r_{1},r_{2} \) are the two records for which the similarity is being calculated.

\( W_{i} \) is the weight of field \( i \).

\( V_{i} \) is the similarity vector of two records \( r_{1},r_{2} \) of field \( i \)

Two records \( r_{1},r_{2} \) are duplicates if similarity \( (r_{1},r_{2}) \geq T_{sim} \), i.e, if the similarity value equal to or greater than the similarity threshold (user defined value to indicate duplicates) in general the similarity \( T_{sim} \) should be closed one. To ensure that the identified duplicates are correct increasing the
value of $T_{sim}$ is reduce the number of duplicate vectors identified by the classifier one while. At the same time the identified duplicate will be more precise.

### 2.6 Support Vector Machine (SVM) classifier

It is a tool used to classify data. SVM uses two steps process. They are training and classifier. In the training step labelled data is supplied to the classifier. Labelling each records has either positive or negative. SVM internally put the information by separating the data into groups. During the classification steps when the system is spied with data to be classified. It classifies the record as either positive or negative based on the training data.

The WCSS classifier outputs three sets of similarity vectors namely potential duplicate vector, non duplicate vector and identified duplicate vectors. From these vectors the identified duplicate vectors D are send as positive examples and non duplicate vector N as negative example for training purpose. These two vectors serve as input (training data) to the SVM classifiers and use this trained classifier to identify new duplicate vectors from the potential duplicate vector P.

### 2.7 Training/Learning

Training data for SVM classifier is the similarity vectors from duplicates and non duplicates. Duplicates are labelled as positive and non duplicates as negative.

<table>
<thead>
<tr>
<th></th>
<th>1:1.000</th>
<th>2:0.8000</th>
<th>3:1.000</th>
<th>4:1.000</th>
<th>5:0.000</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>1:0.5</td>
<td>2:1.000</td>
<td>3:1.000</td>
<td>4:1.000</td>
<td>5:0.000</td>
</tr>
<tr>
<td>-1</td>
<td>1:1.000</td>
<td>2:0.000</td>
<td>3:1.000</td>
<td>4:0.000</td>
<td>5:0.000</td>
</tr>
<tr>
<td>-1</td>
<td>1:1.000</td>
<td>2:0.000</td>
<td>3:1.000</td>
<td>4:0.000</td>
<td>5:0.000</td>
</tr>
</tbody>
</table>

### 2.8 Classification

Once training is complete similarity vectors from potential duplicate dataset are send to the SVM classifiers classifies the function to determine if they are duplicates. The output from the classify function is either positive value (Duplicate) or negative value (non duplicate).

### III. EXPERIMENTS

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

We tested UDD on five data sets. To compare UDD with existing approaches.

**Table: Properties of the Dataset**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Entities</th>
<th>Number of Records</th>
<th>Number of Fields</th>
<th>Number Of Sources</th>
<th>Max number of duplicates for an entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>194</td>
<td>1878</td>
<td>16</td>
<td>119</td>
<td>119</td>
</tr>
<tr>
<td>Book-full</td>
<td>840</td>
<td>4356</td>
<td>4</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>Book-titau</td>
<td>540</td>
<td>2659</td>
<td>2</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>Hotel</td>
<td>780</td>
<td>3896</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Movie</td>
<td>156</td>
<td>587</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

Evaluation Metric:

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

**Precision** = #of Correctly Identified Duplicate Pairs / #of All Identified Duplicate pair

**Recall** = #of Correctly Identified Duplicate Pairs / #of True Duplicate Pairs
Thus, we also use the F-measure, to evaluate the classification quality:
\[
F_{\text{measure}} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

IV. EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th>Table: Performance of UDD on the web database datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Book-full</td>
</tr>
<tr>
<td>Book-titau</td>
</tr>
<tr>
<td>Hotel</td>
</tr>
<tr>
<td>Movie</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance comparison between UDD and other Learning Methods on the Book-Full dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>UDD</td>
</tr>
<tr>
<td>SVM</td>
</tr>
<tr>
<td>OSVM</td>
</tr>
<tr>
<td>PEBL</td>
</tr>
<tr>
<td>Christen</td>
</tr>
</tbody>
</table>

Performance of UDD is close to SVM
OSVM is totally fails
UDD is slower than SVM and OSVM because UDD needs two iterations to identify the duplicates.

V. 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 pre-trained 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, 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 indentify the duplicate pairs from all potential duplicate pairs iteratively. Experimental results show that the approach is comparable to previous work data requires training example for identifying duplicates from the query results of multiple Web databases.

REFERENCES
