Decision Tree Formation and Fuzzy Similarity Matching for Duplicates Detection

. S.Sindhu,
Mphil Scholar,
Dept of Computer Science,
Dr. SNS Rajalakshmi College of Arts and Science,
Coimbatore-49,
India.

Ms.R.Suganya,
Asst Professor,
Dept of Computer Science,
Dr. SNS Rajalakshmi College of Arts and Science,
Coimbatore-49,
India.

ABSTRACT

The duplicate detection is the most important process needed in world to find the duplicates. Several algorithms fail to detect the accurate duplicate in the hierarchal data. Our method to detect duplicates is the combination of Decision Tree and Fuzzy Similarity Matching.

Duplicate detection, which is an important subtask of data cleaning, is the task of identifying multiple representations of a same real-world object.

In this paper is discussed to detect duplicate using the Decision tree is used to form the tree from the given input variables. Fuzzy Similarity Matching is used to match the variables and provide the probability value for the matching .Our goals are either on improving the quality of the detected duplicates (effectiveness) or on saving computation time (efficiency).

Keywords: Decision tree induction, xml duplicate detection, conditional probabilities, fuzzy matching.

1. INTRODUCTION

Duplicate detection is a nontrivial task is the fact that duplicates are not exactly equal, often due to errors in the data. Consequently, we cannot use common comparison algorithms that detect exact duplicates.

Instead, we have to compare all object representations, using a possibly complex matching strategy, to decide if they refer to the same real-world object or not.

Detecting duplicates is problems with a long tradition in many domains. The problems are first define a suitable similarity measure, and second efficiently apply the measure to all pairs of objects. With the advent and pervasion of the XML data model, it is necessary to find new similarity measures and to develop efficient methods to detect duplicate elements in nested XML data.

Duplicate Detection is the problem of detecting different entries in a data source representing the same real-world entity.

While Research abounds in the realm of duplicate detection in relational data, there is yet little work for duplicates in other, more complex data models, such as XML.

For instance, within XML data, XML elements may lack any text. However, the hierarchical relationships with other elements potentially provide enough information for meaningful comparisons.

Automatically detecting duplicates is difficult: First, duplicate representations are usually not identical but slightly differ in their values. Second, in principle all pairs of records should be compared, which is infeasible for large volumes of data.

This lecture examines closely the two main components to overcome these difficulties:

(i) Similarity measures are used to automatically identify duplicates when comparing two records. Well-chosen similarity measures improve the *effectiveness* of duplicate detection.

(ii) Algorithms are developed to perform on very large volumes of data in search for duplicates. Well-designed algorithms improve the *efficiency* of duplicate detection. Finally, we discuss methods to evaluate the success of duplicate detection.

Duplicate detection is an expensive operation of disk-based model checkers. It consists of comparing some potentially new states, the candidate states, to previous visited states. Duplicate detection is the process of identifying multiple representations of a same real-world object in a data source.

Duplicate detection is a problem of critical importance in many applications, including customer relationship management, personal information management or data mining.

The problem of XML duplicate detection is particularly tackling in applications like catalog integration or online data cleansing.

Numerous approaches both for relational and XML data exist. In particular for the first goal, the “goodness” of an approach is usually evaluated based on experimental studies.

Although some methods and data sets have gained popularity, it is still difficult to compare different approaches or to assess the quality of one own’s approach.

In existing system, it first present a probabilistic duplicate detection algorithm for hierarchical data called XML Dup. This algorithm considers both the similarity of attribute contents and the relative importance of descendant elements, with respect to the overall similarity score.

The XML Dup system first proposed uses a Bayesian Network model (BN) for XML duplicate detection.

It first present how to construct a Bayesian Network model for duplicate detection, and then show how this model is used to compute the similarity between XML object representations. Given this similarity, we classify two XML objects as duplicates if it is above a given threshold.

The proposed method for the effective duplicate detection is through Decision Tree Formation and Fuzzy Similarity Matching.

Fuzzy matching is an advanced mathematical process that determines the similarities between data, information, and facts, where the outcome is neither true nor false, or 100 percent certain, hence the word, “fuzzy.” The hierarchical and semi-structured nature of XML strongly differs from the flat and structured relational model, which has received the main attention in duplicate detection so far.

The input data is first processed by Decision Tree learning method and then that tree structure is compared by Fuzzy Similarity Matching and results in accurate duplicate detection.

2. DECISION TREE INDUCTION

Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables.

Decision tree induction is one of the basic techniques for data classification. A Decision tree is a flow-chart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf node represent classes or class distributions, The top most node in a tree is the root node.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node that holds the class prediction for that sample. Decision trees can easily be converted to classification rules.

When decision trees are built, many of the branches may reflect noise or outliers in training data. Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

In the prepruning approach, a tree is pruned by halting its construction early. Upon halting the node becomes a leaf. The leaf may hold the most frequent class among the subset samples or the probability distribution of those samples.

The knowledge represented in decision trees can be extracted and represented in the form of classification IF-THEN rules. One rule is created for each path from the root to a leaf node. Each attribute-value pair along a given path forms a conjunction in the rule antecedent. The leaf node holds the class prediction, forming the rule consequent. The IF-THEN rules may be easier for humans to understand, particularly if the given tree is very large.

The basic algorithm for Decision tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner.

2.1Strengths and Weakness of Decision Tree Methods

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

2.2 The weaknesses of decision tree methods

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.

- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.
- Decision trees do not treat well non-rectangular regions. Most decision-tree algorithms only examine a single field at a time. This leads to rectangular classification boxes that may not correspond well with the actual distribution of records in the decision space.

Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

The estimation criterion in the decision tree algorithm is the selection of an attribute to test at each decision node in the tree. The goal is to select the attribute that is most useful for classifying examples. A good quantitative measure of the worth of an attribute is a statistical property called *information gain* that measures how well a given attribute separates the training examples according to their target classification. This measure is used to select among the candidate attributes at each step while growing the tree.

Practical issues in learning decision trees include determining how deeply to grow the decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handing attributes with differing costs, and improving computational efficiency. Avoiding over-fitting the data

2.3 Avoiding over-fitting the data

Over-fitting is a significant practical difficulty for decision tree learning and many other learning methods. There are several approaches to avoiding over-fitting in decision tree learning. These can be grouped into two classes:

a. Approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data.
b. Approaches that allow the tree to over-fit the data and then post prune the tree.

Although the first of these approaches might seem more direct, the second approach of post-pruning over-fit trees has been found to be more successful in practice. This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree.

Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size.

Approaches include:

- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or

pruning) a particular node is likely to produce an improvement beyond the training set.

- Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is based on a heuristic called the Minimum Description Length principle.

The first of the above approaches is the most common and is often referred to as training and validation set approach. In this approach, the available data are separated into two sets of examples: a *training set*, which is used to form the learned hypothesis, and a separate *validation set*, which is used to evaluate the accuracy of this hypothesis over subsequent data and, in particular, to evaluate the impact of pruning this hypothesis.

The problem of identifying duplicate records has been considered under many different names, such as record linkage, merge/purge, entity identification, and object matching. Typically, standard string similarity metrics such as edit distance or vector-space cosine similarity are used to determine whether two values or records are alike enough to be duplicates.

Several problems arise in the context of data integration, where data from distributed and heterogeneous data sources is combined. One of these problems is the possibly inconsistent representation of the same real-world object in the different data sources. When combining data from heterogeneous sources, the ideal result is a unique, complete, and correct representation for every object. Such data quality can only be achieved through data cleansing, where the most important task is to ensure that an object only has one representation in the result. This requires the identification of duplicate objects, and is referred to as object identification or duplicate detection.

Decision tree is a classifier in the form of a tree structure (see Figure 1), where each node is either:

- A leaf node - indicates the value of the target attribute (class) of examples, or
- A decision node - specifies some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test.

A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node, which provides the classification of the instance.

A decision tree consists of 3 types of nodes:

1. Decision nodes - commonly represented by squares
2. Chance nodes - represented by circles
3. End nodes - represented by triangles

Advantages of Decision trees:

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.
- Possible scenarios can be added
- Worst, best and expected values can be determined for different scenarios

- Use a white box model. If a given result is provided by a model.

Disadvantages of decision trees:

- For data including categorical variables with different number of levels, information gain in decision trees are biased in favor of those attributes with more levels.
- Calculations can get very complex particularly if many values are uncertain and/or if many outcomes are linked.

Once the relationship is extracted, then one or more decision rules can be derived that describe the relationships between inputs and targets. Rules can be selected and used to display the decision tree, which provides a means to visually examine and describe the tree-like network of relationships that characterize the input and target values. Decision rules can predict the values of new or unseen observations that contain values for the inputs, but might not contain values for the targets.

Each rule assigns a record or observation from the data set to a node in a branch or segment based on the value of one of the fields or columns in the data set.1 Fields or columns that are used to create the rule are called inputs. Splitting rules are applied one after another, resulting in a hierarchy of branches within branches that produces the characteristic inverted decision tree form.

Each segment or branch is called a node. A node with all its descendent segments forms an additional segment or a branch of that node. The bottom nodes of the decision tree are called leaves (or terminal nodes). For each leaf, the decision rule provides a unique path for data to enter the class that is defined as the leaf. All nodes, including the bottom leaf nodes, have mutually exclusive assignment rules; as a result, records or observations from the parent data set can be found in one node only. Once the decision rules have been determined, it is possible to use the rules to predict new node values based on new or unseen data. In predictive modeling, the decision rule yields the predicted value.

Decision trees are a simple, but powerful form of multiple variable analysis. They provide unique capabilities to supplement, complement, and substitute for

• Traditional statistical forms of analysis (such as multiple linear regressions)

• A variety of data mining tools and techniques (such as neural networks)

• Recently developed multidimensional forms of reporting and analysis found in the Field of business intelligence Decision trees are produced by algorithms that identify various ways of splitting a data set into branch-like segments.

These segments form an inverted decision tree that originates with a root node at the top of the tree. The object of analysis is reflected in this root node as a simple, one-dimensional display in the decision tree interface. The name of the field of data that is the object of analysis is usually displayed, along with the spread or distribution of the values that are contained in that field.

This Decision tree induction is a typical inductive approach to learn knowledge on classification. The key requirements to do mining with decision trees are:

a. Attribute-value description: object or case must be expressible in terms of a fixed collection of properties or attributes. This means that we need to discretize continuous attributes, or this must have been provided in the algorithm.

b. Predefined classes (target attribute values): The categories to which examples are to be assigned must have been established beforehand (supervised data).
c. Discrete classes: A case does or does not belong to a particular class, and there must be more cases than classes.
d. Sufficient data: Usually hundreds or even thousands of training cases.

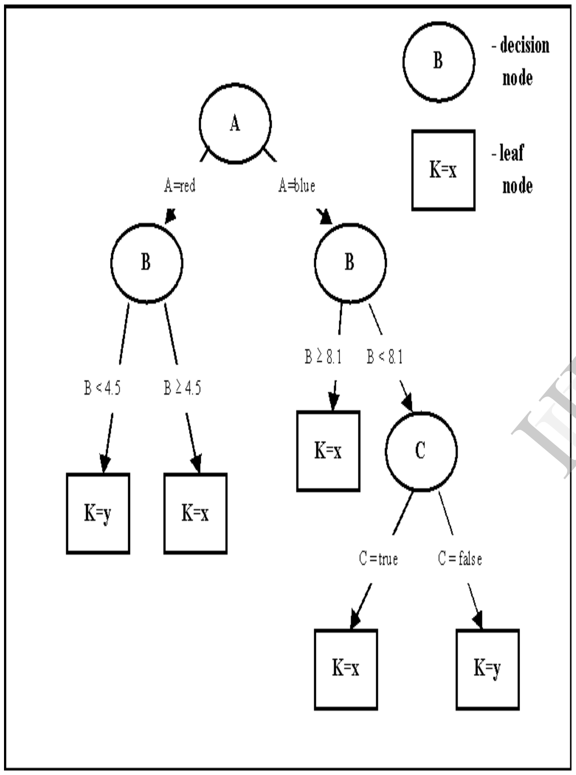


Figure 1: An example of a simple decision tree.

The problem has been addressed extensively for relational data stored in tables. However, relational data only represents a small portion of today's data. Indeed, XML is increasingly popular as data representation, especially for data published on the World Wide Web and data exchanged between organizations.

Therefore, we need to develop methods to detect duplicate objects in nested XML data. There are two types of data heterogeneity: structural and lexical.

Structural heterogeneity occurs when the fields of the tuples in the database are structured differently in different databases. For example, in one database, the customer address might be recorded in one field named, say, addr, while, in another database, the same information might be stored in multiple fields such as street, city, state, and zip code.

Lexical heterogeneity occurs when the tuples have identically structured fields across databases, but the data use different representations to refer to the same real-world object (e.g., Street Address = 44 W. 4th St. versus Street Address = 44 West Fourth Street).

As numerous approaches exist both for increasing efficiency and effectiveness, it is essential to provide some common ground to compare these algorithms with each other.

The proposed algorithms most often improve either efficiency or effectiveness. In the first case, the goal is to reduce the number of pairwise comparisons, which is quadratic in the number of elements if all pairs are compared.

Many duplicate detection algorithms are described in scientific papers only, and often a 12 page publication cannot cover all details and aspects of an approach. When it comes to re-implementing an existing method, the information provided in a paper is often insufficient.

Freely available and simultaneously interesting datasets for duplicate detection are rare. Even more seldom are datasets with true duplicates already marked. As a consequence, even if same or similar datasets were used, the results expressed as precision, recall and run time measure are not comparable: Two approaches might not agree in what is a correctly detected duplicate and how many duplicates are in fact hidden in the dataset.

To confront these problems, many publications create their own data sets, inject duplicates and then let their duplicate detection algorithm find them. Often these generators of data are not freely available for legal reasons simple because the original code is lost once the programmer leaves the organization.

By applying different duplicate detection approaches on the same data, comparing efficiency or effectiveness of different approaches is easy.

Duplicate elements are created s copies of elements in the clean data, Duplicate detection can be considered as batch process, where all pairs of duplicates are determined in single process or it can be considered a search problem, i.e., given a particular element , find its duplicates in a given data set.

The latter is particularly important to support during data input, so that users can warned of a possibly existing entry for the particular real-world object.

Numerous approaches both for relational and xml data exist. Their goals are either on improving the quality of the detected duplicates or on saving computation time. In particular for the first goal, the goodness of the approach is usually evaluated based on experimental studies. Although some methods and data sets have gained popularity it is still difficult to compare different approaches or to assess the quality of one own's approach.

The goal our algorithm is to improve efficiency without reducing effectiveness. When concentrating on effectiveness, the goal is to find duplicates more accurately.

Structure. This paper is organized as follows: Section 3presents XML Duplicate Detection. Section4 summarizes Conditional Probabilities. Section 5.We discusses the results and algorithm .Finally, in Section 6we conclude and present suggestions for future work. This XML duplicate detection is to develop effective, efficient, and scalable solutions for XML duplicate detection.

3. XML DUPLICATE DETECTION:

Research on fuzzy duplicate detection has mainly concentrated on efficiently and effectively finding duplicate records in relational data. Due to space limitations, we only highlight a few solutions in this section. We classify approaches into two areas: domain dependent solutions, such as the ones proposed in, assume a certain domain and the help of human experts to calibrate algorithms.

Domain dependent approaches have in common that description selection and structural heterogeneity is not considered because the problems do not arise in relational duplicate detection.

The work presented in detects duplicates in hierarchically organized relations by considering data of tables related to the object table through foreign keys. In this context, instance heterogeneity has to be considered but description selection and schematic heterogeneity are still not an issue.

There is work on identifying similar hierarchical data and XML data. However, most work does not consider the effectiveness of the similarity join. Rather, the authors concentrate on fast execution of the algorithm.

The focus is on the efficient incorporation of tree edit distance in a framework performing approximate XML joins. The authors present upper and lower bounds for the tree edit distance, which are used as filters to avoid expensive tree edit distance computations. They further introduce a sampling method to effectively reduce the amount of data examined during the join operation.

The only approach we are aware of that considers recall and precision of XML similarity joins. They present four different strategies to define the similarity function using the vector space model.

Knowing Object Descriptions is necessary for duplicate detection, because they represent the information that is used to classify objects as duplicates or non-duplicates. Ideally, an Object Descriptions includes information that characterizes a particular object, such as a book's ISBN, and does not vary depending on external influence (e.g., who wrote the book's review).

Fig: 2 workflow diagram

4. CONDITIONAL PROBABILITIES

It needs to define the following four types of conditional probabilities: Conditional probability 1 (CP1): The probability of the values of the nodes being duplicates, given that each individual pair of values contains duplicates.

Intuitively,

1) If all attribute values are duplicates, we consider the XML node values as duplicates.

2) If none of the attribute values are duplicates, we consider the XML node values as non-duplicates;

3) If some of the attribute values are duplicates, we determine that the probability of the XML nodes being duplicates equals a given value.

This value represents the importance of the corresponding attribute a in determining if the nodes are duplicates.

Conditional probability 2 (CP2): The probability of the children nodes being duplicates, given that each individual pair of children is duplicates. Intuitively, it makes sense to say that two nodes are duplicates only if all of their child nodes are also duplicates.

However, it may be the case that the XML tree is incomplete, or contains erroneous information. Thus, we relax this assumption and state that the more child nodes in both trees are duplicates, the higher the probability that the parent nodes are duplicates.

Conditional probability 3 (CP3): The probability of two nodes being duplicates given that their values and their children are duplicates. Essentially, we consider the nodes as duplicates if both their values and their children are duplicates.

Conditional probability 4 (CP4): The probability of a set of nodes of the same type being duplicates given that each pair of individual nodes in the sets is duplicates. Similarity scores are calculated using the probability values.

5. FUZZY MATCHING

Fuzzy set theory defines fuzzy operators on fuzzy sets. Fuzzy logic needs to be able to manipulate degrees of maybe, in addition to true and false.”

This algorithm combination will result in accurate duplicate detection than the other methods.

The problem in applying this is that the appropriate fuzzy operator may not be known. For this reason, fuzzy logic usually uses IF-THEN rules, or constructs that are equivalent, such as fuzzy associative matrices. Rules are usually expressed in the form:

IF *variable* IS *property* THEN *action*

For example, a simple temperature regulator that uses a fan might look like this:

IF temperature IS very cold THEN stop fan
IF temperature IS cold THEN turn down fan
IF temperature IS normal THEN maintain level
IF temperature IS hot THEN speed up fan

There is no "ELSE" – all of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to different degrees.

The AND, OR, and OT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the *Zadeh operators*.

So for the fuzzy variables x and y:

NOT x = (1 - truth(x))
x AND y = minimum(truth(x), truth(y))
x OR y = maximum(truth(x), truth(y))

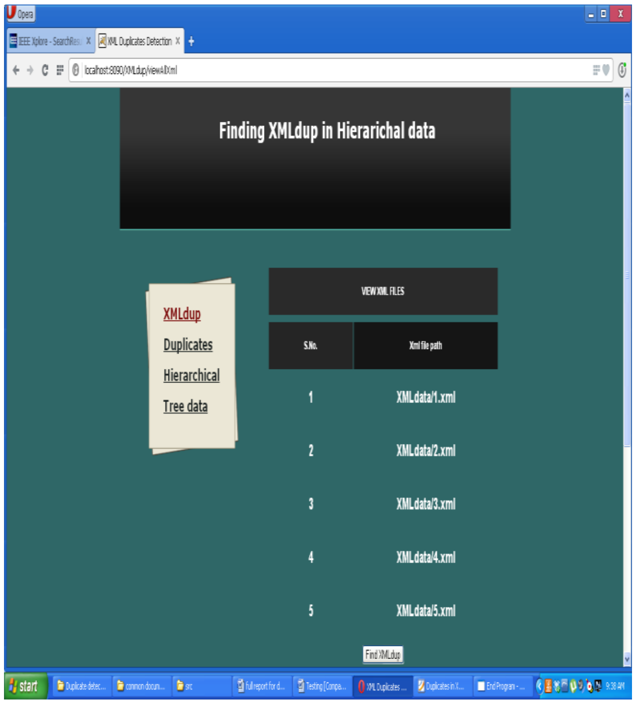
There are also other operators, more linguistic in nature, called *hedges* that can be applied. These are generally adverbs such as "very", or "somewhat", which modify the meaning of a set.

Algorithm:

- Input as records, IR1, IR2,….IRn
- Target records for comparison, TRi1, TRi2,….TRin, where i = 1,2,….,m, where m is the number of records in dataset
- Assume some attributes as important for comparison, for eg., IR1, IR2, IRn-1 &IRn as important attribute.
- For i = 1,2,….,m

a. Similarity score calculation, S1 = PrDiff(IR1,TRi1), S2 = PrDiff(IR2,TRi2) ……… Sn = PrDiff(IRn,TRin)
b. Total similarity score (S) = S1 + S2 + ……. + Sn
c. If S < threshold; Result as Duplicates.
d. Fuzzy match, Calculate S1(IR1,TRi1) AND S2(IR2,TRi2) OR S3(IR3,TRi3) AND …. Sn-2(IRn-2,TRin-2) OR Sn-1(IRn-1,TRin-1) AND Sn(IRn-1,TRin-1)
e. If any Similarity score > 0, Result as

Not Duplicates.

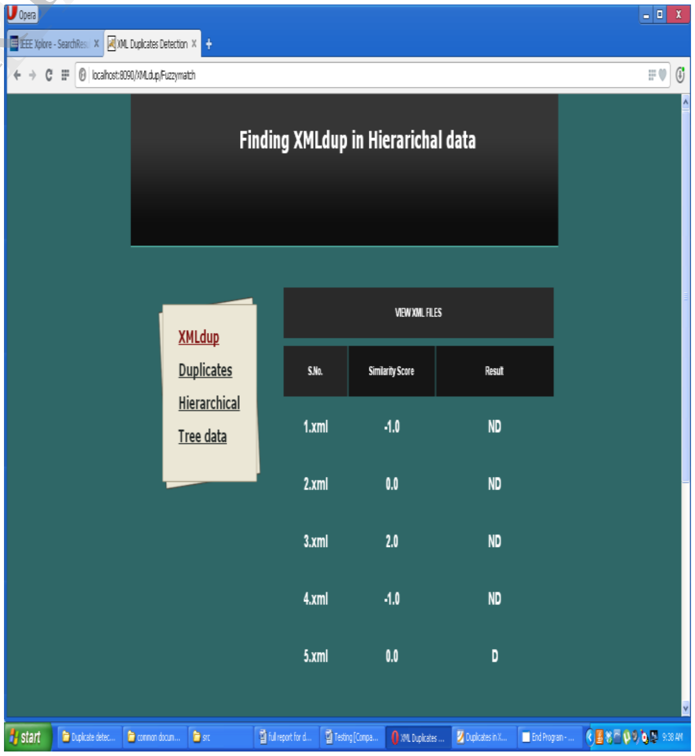


Finding XMLdup in Hierarichal data

VIEW XML FILES

S.No.	Xml file path
1	XMLdata/1.xml
2	XMLdata/2.xml
3	XMLdata/3.xml
4	XMLdata/4.xml
5	XMLdata/5.xml

Fig: 3 List of XML Files



Finding XMLdup in Hierarichal data

VIEW XML FILES

S.No.	Similarity Score	Result
1.xml	-1.0	ND
2.xml	0.0	ND
3.xml	2.0	ND
4.xml	-1.0	ND
5.xml	0.0	D

Fig: 4 Finding duplicates in XML Data

6. CONCLUSION

Duplicate detection is defined as the identification of different representations or versions of a same real-world object, called duplicates.

Such duplicates are due to errors and inconsistencies in the data, such as typos and misspellings, missing information, or outdated data. As a consequence, duplicates are not exactly equal, which makes duplicate detection a challenging task in data cleaning and data integration processes.

Errors are 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.

Many times, the same logical real world entity may have multiple representations in the data warehouse.

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 Super Mart customers in Seattle), and erroneous data mining models to be built.

Duplicate detection is hard because it is caused by several types of errors like typographical errors, and *equivalence errors*—different (non-unique and nonstandard) representations of the same logical value. For instance, a user may enter "WA, United States" or "Wash., USA" for "WA, United States of America."

Equivalence errors in product tables ("winxp pro" for "windows XP Professional") are different from those encountered in bibliographic tables ("VLDB" for "very large databases"), etc. Also, it is important to detect and clean equivalence errors because an equivalence error may result in several duplicate tuples.

Fuzzy matching process will give a probability score to determine the accuracy of the match. This matching will shows the accurate result than Bayesian network.

While considering the problem of XML duplicates detection under the aspects of effectiveness, efficiency and scalability, we believe that our solutions will significantly contribute to solving XML duplicate detection for a wide range of applications.

7. REFERENCES

[1] E. Rahm and H.H. Do, "Data Cleaning: Problems and Current Approaches," IEEE Data Eng. Bull., vol. 23, no. 4, pp. 3-13, Dec. 2000.

[2] F. Naumann and M. Herschel, An Introduction to Duplicate Detection. Morgan and Claypool, 2010.

[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti, "Eliminating Fuzzy Duplicates in Data Warehouses," Proc. Conf. Very Large Databases (VLDB), pp. 586-597, 2002.

[4] D.V. Kalashnikov and S. Mehrotra, "Domain-Independent Data Cleaning via Analysis of Entity-Relationship Graph."ACM Trans. Database Systems, vol. 31, no. 2, pp. 716-767, 2006.

[5] M. Weis and F. Naumann, "Dogmatix Tracks Down Duplicates in XML," Proc. ACM SIGMOD Conf. Management of Data, pp. 431-442, 2005.

[6] A.M. Kade and C.A. Heuser, "Matching XML Documents in Highly Dynamic Applications," Proc. ACM Symp. Document Eng. (DocEng), pp. 191-198, 2008.

[7]S.Puhlmann, M.Weis, and F. Naumann. **Xml duplicate detection** using sorted neigborhoods. International Conference on Extending Database Technology (EDBT), 2006

[8] "A Duplicate Detection Benchmark for XML (and Relational) Data." Melanie Weis and Felix Naumann and FranziskaBrosy, SIGMOD Workshop on Information Quality in Information Systems (IQIS), 2006

[9] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In International Conference on Very Large Databases (VLDB), Hong Kong, China, 2002

[10] E. Cesario, F. Folino, G. Manco, and L. Pontieri.An incremental clustering scheme for duplicate detection in large databases. In Proceedings of the International Database Engineering Application Symposium (IDEAS), pages 89–95, Montreal, Canada, 2005

[11]"xml duplicate detection using sorted neighborhoods"Melanie Weis, Felix Naumann and FranziskaBrosy,SIGMOD 2006 Workshop on Information Quality for Information Systems (IQIS), Chicago, IL, 2006.

[12]"Relationship-Based Duplicate Detection" ,Melanie Weis and Felix Naumann,Technical Report No. HU-IB-206, July 2006.