An Efficient way of Record Linkage System and Deduplication using Indexing techniques, Classification and FEBRL Framework

Nishand.K[1], Ramasami.S[2], Dr. T.Rajendran[3]
[1]II-ME CSE, Department of Computer Science and Engineering,
[2]Assistant Professor, Department of Computer Science and Engineering
[3]Dean, Department of Computer Science and Engineering
[1,2,3]Angel College of Engineering and Technology, Tirupur, India
[2]sramasami)@gmail.com,[3]rajendran_tm@yahoo.com

***Abstract*: Record linkage is an important process in data integration, which is used in merging, matching and duplicate removal from several databases that refer to the same entities. Deduplication is the process of removing duplicate records in a single database. In recent years, data cleaning and standardization becomes an important process in data mining task. Due to complexity of today's database, finding matching records in single database is a crucial one. Indexing techniques are used to efficiently implement record linkage and deduplication. In this paper, three indexing techniques namely blocking index, sorting indexing and bigram indexing are used with a modification of existing techniques that reduces the variance in the quality of the blocking results. In addition to the indexing techniques, six comparison techniques and two classifiers are used. There is a potential for large performance speed-ups and better accuracy to be achieved by using indexing techniques along with comparison and classifier techniques.**

***Keywords:* Record linkage, Indexing techniques, data matching, blocking, Febrl framework**

1. INTRODUCTION

With many businesses, government agencies and research projects collecting massive amounts of data, techniques that allow efficient processing, analyzing and mining of large databases have in recent years attracted interest from both academia and industry. An increasingly important task in the data preparation phase of many data mining projects is linking or matching records relating to the same entity from several databases, as often information from multiple sources needs to be integrated and combined in order to enrich data and allow more detailed data mining studies. The aim of such linkages is to match and aggregate all records relating to the same entity, such as a patient, a customer, a business, a consumer product, a bibliographic citation, or a genome sequence.

Record linkage and deduplication can be used to improve data quality and integrity, to allow re-use of existing data sources for new studies, and to reduce costs and efforts in data acquisition. Record linkage can also help to enrich data that is used for pattern detection in data mining systems. Businesses routinely deduplicate and link their data sets to compile mailing lists, while within taxation offices and departments of social security, record linkage and deduplication can be used to identify people who register for benefits multiple times or who work and collect unemployment money. Another application of current interest is the use of data linkage in crime and terror detection. Security agencies and crime investigators increasingly rely on the ability to quickly access files for a particular individual, which may help to prevent crimes by early intervention. The problem of finding records that relate to the same entities not only applies to databases that contain information about people. Other types of entities that sometimes need to be matched include records about businesses, consumer products, publications, and bibliographic citations, web pages, web search results, or genome sequences. In bioinformatics, for example, record linkage techniques can help find genome sequences in large data collections that are similar to a new, unknown sequence. In the field of information retrieval, it is important to remove duplicate documents (such as web pages and bibliographic citations) in the results returned by search engines, in digital libraries or in automatic text indexing systems. Another application of growing interest is finding and comparing consumer products from different online stores. Because product descriptions are often slightly varying, matching them becomes challenging.

Removing duplicate records in a single database is a crucial step. Deduplication can be achieved more efficiently by using indexing techniques. One or more (blocking) indexes need to be built with the aim of grouping together records that potentially match and thus reducing the huge number of possible comparisons. While this grouping should reduce the number of comparisons made as much as possible, it is important that no potential match is overlooked because of the indexing process. After index are built, records within the same index block are compared by using field comparison functions, resulting in a weight vector for each record pair compared. These weight vectors are then given to a classifier that decides if a record pair constitutes a match, non-match or a possible match. In this paper Blocking method, Sort Indexing and Bigram indexing are used with changes in the existing system. In addition to indexing technique, six comparison and two classifiers are used to increase the efficiency.

1.1 RECORD LINKAGE PROCESS

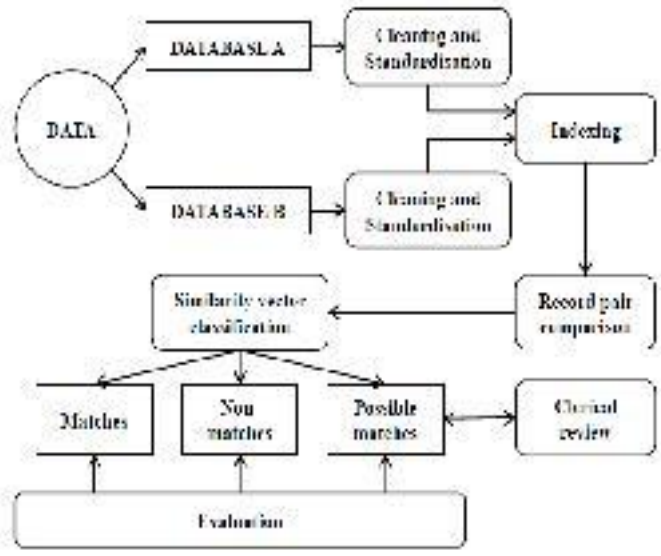

Fig 1. Outline process of Record linkage

Record linkage techniques are used to link data records relating to the same entities, such as patients or customers. Fig.1 shows the outline of record linkage. Record linkage can be used to improve data quality and integrity, to allow re-use of existing data sources for new studies, and to reduce costs and effort in data acquisition for research studies. If a unique entity identifier or key is available in all of the data sets to be linked, then the problem of linking at the entity level is trivial – a simple join operation in SQL or its equivalent in other data management systems is all that is required. However, if no unique key is shared by all of the data sets, then various record linkage techniques need to be used. No matter what technique is used, a number of issues need to be addressed when linking data. Often, data is recorded or captured in various formats, and data items may be missing or contain errors. A pre-processing phase that aims to clean and standardize the data is therefore an essential first step in every linkage process. Data sets may also contain duplicate entries, in which case linkage may need to be applied within a data set to be de-duplicate it before linkage with other files is attempted. The process of linking records has various names in different user communities. While epidemiologists and statisticians speak of record linkage, the process is often referred to as data matching or as the object identity problem by computer scientists, whereas it is sometimes called merge/purge processing or list washing in commercial processing of customer databases or mailing lists. Historically, the statistical and the computer science communities have developed their own techniques, and until recently few cross-references could be found. Computer-assisted record linkage goes back as far as the 1950s. At this time, most linkage projects were based on adhoc heuristic methods. The basic ideas of probabilistic record linkage were introduced by Newcombe and Kennedy in 1962 while the theoretical foundation was provided by Fellegi and Sunter in 1969. Using frequency counts to derive agreement and disagreement probabilities, each pair of fields of each compared record pair is assigned a match weight, and critical values of the sum of these match weights are used to designate a pair of records as either a link, a possible or anon-link. Possible links are those pairs for which human oversight, also known as clerical review, is needed to decide their final linkage status. In theory, the person undertaking this clerical review has access to additional data (or may be able to seek it out) which enables them to resolve the linkage status. In practice, often no additional data is available and the clerical review process becomes one of applying human or common sense to the decision based on available data. One of the aims of the FEBRL project is to automate (and possible improve upon) this process through the use of machine learning and data mining techniques. To reduce the number of comparisons (potentially each record in one data set has to be compared with every record in a second data set), blocking techniques are typically used. The data sets are split into smaller blocks using blocking variables, like the postcode or the Soundex encoding of surnames. Only records within the same blocks are then compared. To deal with typographical variations and data entry errors, approximate string comparison functions are often used for names and addresses. These comparators usually return a score between 0.0 (two strings are completely different) and 1.0 (two strings are the same). The terms data cleaning, data standardization, data scrubbing, data pre-processing and ETL(extraction, transformation and loading) are used synonymously to refer to the general tasks of transforming the source data (often derived from

operational, transactional information systems) into clean and consistent sets of records which are suitable for record linkage of for loading into a data warehouse. The meaning of the term standardization in this context is quite different from its use in epidemiology and statistics, where it usually refers to a method of dealing with the confounding effects of age. The main task of data standardization in record linkage is the resolution of inconsistencies in the way information is represented or encoded in the data. Inconsistencies can arise through typographical or other data capture errors, the use of different code sets or abbreviations and differences in record layouts. The FEBRL System is implemented in an object oriented design with a handful of modules, each containing routines for specific tasks. Record linkage consists of two main steps. The first one deals with data cleaning and standardization, while the second performs the actual linkage (or deduplication). The user needs to specify various settings in order to be able to perform a cleaning/standardization and/or a linkage /deduplication/geocoding process.

2. INDEXING

The aim of indexing is to reduce the potentially huge number of comparisons (every record in one data set with all records in another data set) by eliminating comparisons between records that obviously are not matches. In other words, indexing reduces the large search space by forming groups of records that are very likely to be matches. Indexing can also be seen as a clustering method that brings together records that are similar, so only these records need to be compared using the more expensive (i.e. compute intensive) field comparisons functions.

2.1 BLOCKING INDEXING

Currently the FEBRL system contains several indexing methods, including the traditional blocking method used in many record linkage systems. Indexes are normally built while a data set is being standardized. After an index is built a compacting has to be done which builds index data structures that can return the blocks more efficiently. In the following example, three indexes are defined and a traditional blocking index is initialized. The first index is based on the Soundex encoding of the reversed surname field values, with a maximal code length of three, the second index is based on a combination of the first two characters in the given name field (the truncate method is used for this) concatenated with the values in the postcode field (the method is direct which means the postcode values are not encoded or modified in any way), and the third index is based on concatenation of the first two digits in the postcode plus the NYSIIS encoding of the surname values.

```
hosp_block_def=[[(„surname“,“soundex“,3,“reverse“)],[(„gi
venname“,“truncate“,2),(„postcode“,“direct“)],[(„postcode“,“t
runcate“,2),(„surname“,“nysiis“)],]
hospital_index=BlockingIndex(name=“HospIndex“,
dataset=tmpdata, block_def=hosp_block_def)
```

2.2 SORTING INDEXING

The sorting index extends the idea of the classical blocking index in that the values of the blocks (e.g. the Soundex encodings of surnames) are sorted alphabetically and then a sliding window is moved over these sorted blocks. When a sorting index is initialized, one argument needs to be given is:

Windowing method: A positive integer that gives the size of the sliding window. For example, let's assume there are six blocks in our index (shown are the blocking variable values and the corresponding record numbers in the blocks), and we have a sliding window size of 3:

a123: [4,12,89,99]
a129: [6,32,54,84,91]
a245: [1,39]
a689: [3,17,21,35,49,76,87,93]
a911: [2,42,66]
b111: [8]

While with the blocking index only the records within one block are compared with the records in the corresponding block of the second data set, with the sorting index and window size 3 larger blocks are formed by combining three consecutive blocks together:

[a123,a129,a245]: [1,4,6,12,32,39,54,84,89,91,99]
[a129,a245,a689]:[1,3,6,17,21,32,35,39,49,54,76,84,87,91, 93]
[a245,a689,a911]: [1,2,3,17,21,35,39,42,49,66,76,87,93]
[a689,a911,b111]: [2,3,8,17,21,35,42,49,66,76,87,93]

The idea behind this is, that neighboring blocks might contain records with similar values in the blocking variables due to errors in the original values. Note that if the window size is set to then the sorting index becomes equivalent to the blocking index.

2.3. BIGRAM INDEXING

The aim of this technique is to index the database(s) such that records that have a similar, not just the same, BKV will be inserted into the same block. The basic idea is to create variations for each BKV using bigram, and to insert record identifiers into more than one block. This index implements a data structure based on bigrams and allows for fuzzy blocking. The basic idea is that after an index has been built, the values of the blocking variables will be converted into a list of bigrams, which is sorted alphabetically (and duplicate bigrams are removed), and sub-lists will be built using a user provided threshold (a number between 0.0 and 1.0) of all possible combinations. These resulting bigram sub-lists will be inserted into an inverted index, i.e. record numbers

in the blocks will be inserted into dictionaries for each bigram sub-list. Such an inverted index will then be used to retrieve the blocks.

A blocking key value "baxter" will result in a bigram list ("ba","ax","xt","te","er"). With a threshold of 0.8 the following sub-lists of length 4 (calculated as the length of the bigram list times the threshold: 5×0.8) will be inserted into the inverted index:

("ax","xt","te","er")
("ba","xt","te","er")
("ba","ax","te","er")
("ba","ax","xt","er")
("ba","ax","xt","te")

All record numbers which contain the blocking key value "baxter" will be inserted into five blocks, thus increasing the number of record pair comparisons compared to standard blocking. The number of sub-lists created for a blocking key value both depends on the length of the value and the threshold. The lower the threshold the shorter the sub-lists, but also the more sub-lists there will be per blocking key value, resulting in more (smaller blocks) in the inverted index.

5. RECORD COMPARATOR

A Record Comparator is constructed using a list of field comparators. Once field comparison functions have been initialized, then they can be inserted into a field_comparisons list which is then given to a record comparator. When comparing records in a linkage or deduplication process, each field comparator in a field_comparisons list will calculate a weight, and all weights for a record pair will be stored in a weight vector. Additional information stored with a weight vector is the unique record identifiers for the two records in the pair compared. The weight vectors will then be used to compute the matching status of the record pair in the classifier. The Record Comparator is initialized using references to two data sets to be linked (which can be the same in case of a deduplication task), and a list of field comparison functions. The comparison techniques used are

5.1 EXACT STRING COMPARISON

This field comparator function compares the two fields (or field lists) given to it as strings and returns the agreement weight if they are the same and the disagreement weight if they differ.

5.2 TRUNCATED STRING COMPARISON

This field comparison function allows the comparison of strings that can be truncated at a certain position using the argument max_string_length. Only the first max_string_length characters in both strings are compared. Similar to the exact string comparison function, this field comparator compares the two fields (or field lists) given to it as strings and returns the agreement weight if the truncated strings are the same and the disagreement weight if they differ.

5.3 APPROXIMATE STRING COMPARISON

Approximate string comparison is an important feature for successful weight calculation when comparing strings from names and addresses. Instead of simply having an agreement or disagreement weight returned, approximate string comparators allow for partial agreement if strings are not exactly but almost the same, which can be due to typographical and other errors. Various algorithms for approximate string comparisons have been developed, in both the statistical record linkage and in the computer science and natural language processing communities. All string comparison functions implemented return a value between (two strings are completely different) and (two strings are the same). The approximate string comparison method has to be selected with the compare_method argument. The following methods are currently implemented:

• jaro

The Jaro string comparator is commonly used in record linkage software. It computes the number of common characters in two strings, the lengths of both strings, and the number of transpositions to compute a similarity measure between and .

• winkler

The Winkler comparator is based on the Jaro comparator but takes into account the fact that typographical errors occur more often towards the end of words, and thus gives an increased value to characters in agreement at the beginning of the strings. The partial agreement weight is therefore increased if the beginning of two strings is the same.

• bigram

Bigrams are the two-character substrings in a string, for example 'peter' contains the bigrams 'pe', 'et', 'te' and 'er'. In the Bigram string comparator, the number of common bigrams in the two strings is counted and divided by the average number of bigrams in the two strings, to calculate a similarity measure between 0.0 and 1.0.

• editdist

The Edit distance algorithm (also known as the Levenshtein distance) counts the minimum number of deletions, transpositions and insertions that have to be made to transform one string into the other. This number is then divided by the length of the longer string to get a similarity measure between 0.0 and 1.0.

• seqmatch

This approximate string comparator is implemented in the Python standard library difflib. It is based on an algorithm developed by Ratcliff and Obershelp in the 1980s, and uses

pattern matching to compute a similarity measure between 0.0 and 1.0.

5.4 ENCODED STRING COMPARISON

Phonetic name encoding is traditionally used to create blocking variables in the record linkage process, but it can also be used to compare strings. The encoded string comparison function compares the two fields (or field lists) given to it as encoded strings, and returns the agreement weight if both strings are encoded the same way, otherwise the disagreement weight is returned.

5.5 KEYING DIFFERENCE COMPARISON

This field comparator compares the two fields (or field lists) given to it as strings character-wise, with a maximum number of different characters that can be tolerated. This number has to be set with the argument max_key_diff. If the number of different characters is larger than zero but equal to or smaller than max_key_diff the partial agreement weight is calculated. If the number of key differences is larger than max_key_diff then the disagreement weight is returned. This field comparator can also be used to compare numerical fields, such as date of birth, telephone numbers, etc.

5.6 NUMERIC COMPARISON WITH PERCENTAGE TOLERANCE

This field comparator is for numeric fields, where a given maximal percentage difference can be tolerated. This percentage has to be set by the argument max_perc_diff as a value between and . The default value is , i.e. no difference is tolerated. The agreement weight is returned if the numbers are the same and the disagreement weight if the percentage difference is larger than the max_perc_diff value. If the percentage difference between the two values is larger than but equal to or smaller than max_perc_diff the partial agreement weight is calculated.

6. CLASSIFICATION

The last step in a record linkage process - after records have been compared and weight vectors have been calculated - is the classification of record pairs into links, non-links, or if this decision should be done by a human review, possible links. Two classifiers used are Fellegi & Sunter classifier and a Flexible classifier.

6.1 FELLEGI AND SUNTER CLASSIFIER

The classical Fellegi and Sunter classifier simply sums all the weights in a weight vector, and then uses two thresholds to classify a record pair into one of the three classes links, non-links or possible links.

6.2 FLEXIBLE CLASSIFIER

This flexible classifier allows different methods to be used to calculate the final matching weight for a weight vector. Similar to the Fellegi and Sunter classifier, two thresholds are used to classify a record pair into one of the three classes links, non-links or possible links. The results of a classification are stored in a data structure. Instead of simply summing all weights in a weight vector, this flexible classifier allows a flexible definition of the final weight calculation by defining tuples containing a function and elements of the weight vector upon which the function is applied. The final weight is then calculated using another function that needs to be defined by the user.

7. EXPERIMENTAL EVALUATION

In previous works indexing techniques are alone used to record linkage and deduplication. In this paper, indexing techniques along with classification and comparators are used. All these are implemented in FEBRL framework. Datasets used are real data and artificially generated data. Because in real datasets it is difficult to identify the deviations in results. So artificial datasets are also used. Artificial data are generated using febrl framework. This generator first creates original records based on frequency tables that contain real name and address values, as well as other personal attributes; followed by the generation of duplicates of these records based on random modifications such as inserting, deleting, or substituting characters, and swapping, removing, inserting, splitting, or merging words. The types and frequencies of these modifications are also based on real characteristics. The true match status of all record pairs is known. The original and duplicate records were then stored into one file each to facilitate their linkage.
Four measures are used to assess the complexity of the indexing step and the quality of the resulting candidate record pairs. The total number of matched and nonmatched record pairs are denoted with nM and nN, respectively, with nM + nM = nA × nB for the linkage of two databases, and nM + nN = nA(nA – 1)/2 for the deduplication of one database. The number of true matched and true nonmatched record pairs generated by an indexing technique is denoted with sM and sN, respectively, with sM + sN ≤ nM + nN.

The reduction ratio, RR = 1:0 - $\frac{s_M + s_N}{n_M + n_N}$

, measures the reduction of the comparison space, i.e., the fraction of record pairs that are removed by an indexing technique. The higher the RR value, the less candidate record pairs are being generated. However, reduction ratio does not take the quality of the generated candidate record pairs into account (how many are true matches or not). Fig. 7.1 shows the evaluation of the given data set.

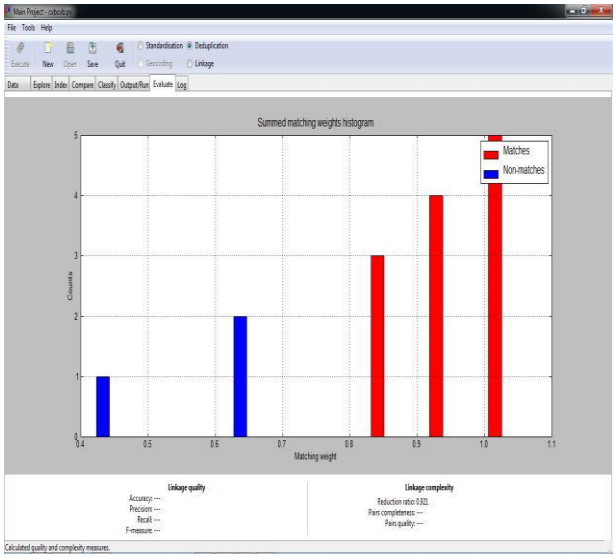

Fig. 7.1 Evaluation page showing the matching weight histogram and quality and complexity measures for a linkage.

8. CONCLUSION AND FUTURE WORK

Record linkage and deduplication are important steps in the pre-processing phase of many data mining projects, and also important for improving data quality before data is loaded into data warehouses. Different record linkage techniques have been presented along with comparators and classifiers, which are used to separate records into match, non-match and possible matches. The indexing techniques presented in this paper are heuristic approaches that aim to split the records in a database (or databases) into (possibly overlapping) blocks such that matches are inserted into the same block and nonmatches into different blocks. While future work in the area of indexing for record linkage and deduplication should include the development of more efficient and more scalable new indexing techniques, the ultimate goal of such research will be to develop techniques that generate blocks such that it can be proven that 1) all comparisons between records within a block will have a certain minimum similarity with each other, and 2) the similarity between records in different blocks is below this minimum similarity.

REFERENCES

[1] T. Churches, P. Christen, K. Lim, and J.X. Zhu, "Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models," BioMed Central Medical Informatics and Decision Making, vol. 2, no. 9, 2002.

[2]. P. Christen, "Febrl: An Open Source Data Cleaning, Deduplication and Record Linkage System With a Graphical User Interface," Proc. 14th ACM SIGKDD Int‟l Conf. Knowledge Discovery and Data Mining (KDD ‟08), pp. 1065-1068, 2008.

[3]. L. Gu and R. Baxter, "Decision Models for Record Linkage," Selected Papers from AusDM, LNCS 3755, Springer, 2006.

[4]. S. Yan, D. Lee, M.Y. Kan, and L.C. Giles, "Adaptive Sorted Neighborhood Methods for Efficient Record Linkage," Proc. Seventh ACM/IEEE-CS Joint Conf. Digital Libraries (JCDL ‟07), 2007.

[5]. 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 (VLDB ‟01), pp. 491-500, 2001.

[6]. J.I. Maletic and A. Marcus, "Data Cleansing: Beyond Integrity Analysis," Proc. Fifth Conf. Information Quality (IQ ‟00), pp. 200- 209, 2000.

[7]. L. Jin, C. Li, and S. Mehrotra, "Efficient Record Linkage in Large Data Sets," Proc. Eighth Int‟l Conf. Database Systems for Advanced Applications (DASFAA ‟03), pp. 137-146, 2003.

[8]. C. Faloutsos and K.-I. Lin, "Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," Proc. ACM SIGMOD Int‟l Conf. Management of Data (SIGMOD ‟95), pp. 163-174, 1995.