Development and User Experiences of an Open Source Data Cleaning, Deduplication and Record Linkage System

Peter Christen
School of Computer Science
The Australian National University
Canberra ACT 0200, Australia
peter.christen@anu.edu.au

ABSTRACT

Record linkage, also known as database matching or entity resolution, is now recognised as a core step in the KDD process. Data mining projects increasingly require that information from several sources is combined before the actual mining can be conducted. Also of increasing interest is the deduplication of a single database. The objectives of record linkage and deduplication are to identify, match and merge all records that relate to the same real-world entities. Because real-world data is commonly 'dirty', data cleaning is an important first step in many deduplication, record linkage, and data mining projects.

In this paper, an overview of the *Febrl* (Freely Extensible Biomedical Record Linkage) system is provided, and the results of a recent survey of *Febrl* users is discussed. *Febrl* includes a variety of functionalities required for data cleaning, deduplication and record linkage, and it provides a graphical user interface that facilitates its application for users who do not have programming experience.

Categories and Subject Descriptors: H.2.8 [Database applications]: Data mining

General Terms: Algorithms, Experimentation

Keywords: data linkage, database matching, data standardisation, open source software, Python, GUI.

1. INTRODUCTION

A crucial requirement for successful data mining projects in many application areas is the linking of records from several heterogeneous databases [10; 23]. Related to record linkage is the deduplication of a single database [16]. Record linkage and deduplication are aimed at matching all records that relate to the same real-world entities. These entities can, for example, be patients, customers, tax payers, travellers, or even businesses, publications, patents, or genome sequences. Record linkage and deduplication can be used to improve data quality and integrity, to allow re-use of existing data for new studies, to enable data analysis and mining at levels of details not otherwise possible, and to reduce costs and efforts in data acquisition.

In the health sector, for example, linked data can contain information that is required to improve health policies, and that traditionally has been collected with time consuming and expensive survey methods. Linked data can also help in health surveillance systems to enrich data that is used for detection of suspicious patterns, such as outbreaks of contagious diseases. Businesses routinely deduplicate and link their databases to compile mailing lists for customer analytics purposes, while taxation offices and departments of social security use record linkage to identify people who register for benefits multiple times, or who work and collect unemployment money at the same time. Another area where record linkage has gained increased interest in recent times is crime and terror detection. Security agencies and crime investigators increasingly rely on the ability to quickly bring up files for a particular individual, which may help to prevent crimes or terrorism by early intervention.

A variety of commercial record linkage and deduplication systems are available. From a user's perspective, the vast majority of these systems are a 'black box', because the details of the technology implemented within the linkage engine of these systems are normally not accessible. Additionally, many of these systems are specialised to a certain domain, for example the linking of business data, or the cleaning or deduplication of customer mailing lists.

For many applications, the record linkage or deduplication process is quite complex, because it involves data from heterogeneous sources, possibly from different domains and collected at different points in time. Therefore, a significant amount of customisation or additional programming is commonly required before a commercial linkage system can be successfully used for a certain application. A record linkage application is often limited by the functionality offered by the commercial linkage system employed.

Record linkage is a complex process that requires the user to understand, up to a certain degree, various technical details. For example, it is important that a user appreciates how approximate string comparison functions work on name and address values, because they will significantly influence the quality of the final matched data. Similarly, understanding the trade-offs of using certain record fields (attributes) in the linkage process, or setting parameters to certain values, is crucial. On one hand a specific parameter choice might result in poor linkage quality, while on the other hand a different choice might result in too many records pairs being compared, thus making a linkage not feasible.

Several smaller record linkage systems are available for free or at affordable prices. However, they are commonly limited in their ability to deal with different types of data, only contain a limited amount of functionality (for example implement only a small number of commonly used string comparison functions), or they can only link small data sets. Large-

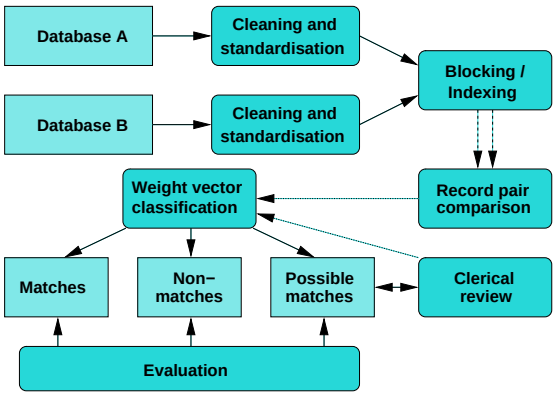

Figure 1: The general record linkage process. The blocking / indexing step generates candidate record pairs, and the output of the record pair comparison step are weight vectors that contain numerical similarity values.

scale record linkage and deduplication systems, on the other hand, are usually very expensive, require powerful computing and large storage environments, and are therefore only affordable by large organisations.

It is important that users of record linkage systems, as well as researchers working in this area, have access to free (or at least affordable) tools that allow them to learn about and experiment with record linkage techniques. Such tools should include both traditional and novel techniques (to allow users to understand their advantages and limitations), they should be flexible and contain a variety of different linkage techniques, and they should allow a multitude of configuration options for a user to conduct a variety of experimental linkages. To facilitate this, the source code of the linkage engine of such systems should be available for inspection, modification and improvement. On the other hand, because many users of record linkage systems have limited programming experience, a graphical user interface should be provided to facilitate the use of record linkage techniques without the need of any programming.

This lack of flexible record linkage systems, that allow access to their source code and include a variety of linkage techniques, is addressed by the *Febrl* (Freely Extensible Biomedical Record Linkage) system presented in this paper. To the best of the author's knowledge, *Febrl* is the only freely available data cleaning, deduplication and record linkage system with a graphical user interface.

This paper is an extended version of a demonstration paper presented at ACM KDD'08 [7]. In the following Section 2, a general overview of the record linkage process is provided. The background of the *Febrl* project is given in Section 3, and in Section 4 the structure and functionality of the *Febrl* software is explained in detail, and illustrated with screenshots of an example record linkage project. Then, in Section 5, the results of a recently conducted survey of *Febrl* users is discussed. The paper is concluded in Section 6 with an outlook to future development plans for *Febrl*.

2. THE RECORD LINKAGE PROCESS

As shown in Figure 1, the linkage process consists of five major steps: cleaning and standardisation, indexing / blocking, comparison, classification, and evaluation / review.

```
Record A:       ['dr',   'peter',   'paul',   'miller']
Record B:       ['mr',   'pete',    '',       'miller']

Weight vector:  [0.5,    0.9,       0.0,      1.0      ]
```

Figure 2: Example weight vector generated when comparing records A and B.

Because common unique entity identifiers (or keys) are often not available in the databases to be linked or deduplicated, the linkage process is usually based on the available record fields (attributes), which in many cases contain personal details such as names, addresses, and dates of birth. The values in these fields, however, often contain noisy, incomplete and incorrectly formatted information. A lack of good quality data can be one of the biggest obstacles to successful record linkage and deduplication [13], because records might not be compared with each other if they contain incorrect or missing information. Data cleaning and standardisation are therefore an important first step for successful record linkage and deduplication. Their objective is the conversion of the raw input data into well defined, consistent formats; and the resolution of inconsistencies in the way information is represented and encoded [12].

When linking two databases, potentially each record in one database needs to be compared with all records in the other database, because there is no way to know prior to the comparison step if two records are matching or not. This comparison process is therefore of quadratic complexity [10]. The maximum number of true matches, however, cannot be larger than the number of records in the smaller of the two databases to be linked. Similarly, when deduplicating a single database, each record potentially needs to be compared with all other records, while the maximum number of potential duplicates is always smaller than the number of records in a database. To improve the scalability of the linkage process, the potentially very large number of record pairs that are to be compared has to be reduced. This can be accomplished through some form of indexing (called 'blocking' in record linkage [2]), that splits the databases into blocks (or clusters). Only records that are in the same block are compared with each other. For example, if a postcode field is used for blocking, then only records that have the same postcode value are compared with each other.

The candidate pairs generated in the blocking step are compared using comparison functions appropriate to the content of the record fields (attributes). Approximate string comparison functions, which take variations and typographical errors into account, are commonly used on name and address fields [3; 14], while comparison functions specific for date, age, and numerical values are used for fields that contain such information. Each comparison function returns a numerical similarity value, often called a *matching weight*, that is commonly normalised, such that 1 corresponds to two exactly matching values, and 0 to two completely different values. Two values that are somewhat similar (such as the two name strings 'Gail' and 'Gayle') will result in a similarity value somewhere between 0 and 1. Several fields of each candidate record pair are normally compared, and a *weight vector* is formed that contains all matching weights calculated for a record pair. Figure 2 shows two example records (made of title, given name, middle name and surname fields) and a corresponding weight vector.

Table 1: Release history of the *Febrl* software and version highlights.

Version	Release date	Highlights
0.1	5 Sep 2002	Data cleaning and standardisation modules only.
0.2	14 Apr 2003	Complete object-oriented re-design, included modules for probabilistic record linkage and deduplication.
0.2.1	25 Jun 2003	Various improvements, and added several new features in existing modules.
0.2.2	20 Nov 2003	Bug-fix release of 0.2.1, no new features were added.
0.3	6 Apr 2005	Added a geocode matching module and various new features in several existing modules, and updated all modules to Python version 2.4.
0.4	7 Nov 2007	Included a GUI, completely re-designed module structure, added many new features, and updated all modules to Python version 2.5.
0.4.01	13 Dec 2007	Bug-fix release of 0.4. A tutorial chapter was added to the manual.
0.4.02	15 Apr 2008	Bug-fix release of 0.4.01.
0.4.1	16 Dec 2008	Included a new, much improved, data generator module [11].

Based on these weight vectors, the next step in the linkage process is to classify the compared candidate record pairs into *matches*, *non-matches*, and *possible matches* (depending upon the decision model used) [10; 19]. Record pairs that were removed in the blocking step are classified as non-matches without being compared explicitly. Various classification techniques have been developed in the past four decades, ranging from basic threshold-based [17; 23] to complex machine learning based approaches [5; 6; 15].

The final step in the linkage process is to evaluate the quality of the generated matches and non-matches. A variety of evaluation measures can be used for this, however, due to the normally imbalanced distribution of matches versus non-matches, care needs to be taken to prevent over-optimistic accuracy results [10]. If a classification technique has been used that classified record pairs as possible matches [23], then a manual clerical review step is required to decide the final linkage status of these record pairs.

3. FEBRL PROJECT BACKGROUND

The *Febrl* software has been developed since early 2002 as part of a collaborative research project conduced by the Australian National University in Canberra and the New South Wales Department of Health in Sydney, Australia. The objectives of this project are to develop novel techniques for improved data cleaning and standardisation, deduplication and record linkage. While the focus of this research is on the cleaning and linking of health related data, the techniques developed and implemented in *Febrl* are general enough to be applicable to data from a variety of other domains.

Since its first publication in early September 2002, the *Febrl* software has been hosted on the *Sourceforge.Net* open source software repository, and it is available from:

`https://sourceforge.net/projects/febrl/`

Febrl is written in the Python[1] programming language. Python provides an ideal platform for rapid prototype development. It includes data structures like sets, lists and associative arrays (called 'dictionaries') that allow efficient handling of very large data sets. It also contains many modules that offer a range of functionalities, including string handling, database access, Web programming, numerical capabilities, and GUI (graphical user interface) development.

[1] `http://www.python.org`

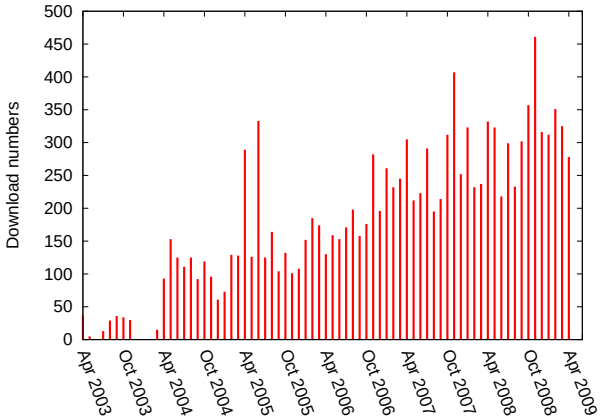

Figure 3: Monthly *Febrl* download numbers from the *Sourceforge.Net* repository. The total number of downloads on 8th May 2009 was 13,234.

Febrl is published under an open source software licence that is based on the Mozilla Public License 1.1[2]. This license allows maximum flexibility for users by permitting them to integrate *Febrl* into other, non open source, software. This is not possible with other open source licenses, such as the GNU General Public License. An overview of the release history of the *Febrl* software can be seen in Table 1. To the best of the author's knowledge, *Febrl* is the only open source software that includes a GUI and allows data cleaning and standardisation, deduplication and record linkage.

As can be seen in Figure 3, the download numbers per month have significantly increased since the initial release of *Febrl*. Note that downloads before April 2003 are not available due to a change in the *Sourceforge.Net* statistics system at that time. Some of the outliers (peak download numbers) coincide with the release of major new *Febrl* versions (0.3 in April 2005 and 0.4 in November 2007), while others (like the peaks in June 2005 and November 2008) are not correlated to any specific event (such as publications or presentations) that increased the publicity of *Febrl*.

Due to the availability of its source code, *Febrl* is suitable for the rapid development, implementation, and testing of novel data cleaning, record linkage and deduplication techniques,

[2] For details see: `http://www.opensource.org/licenses`

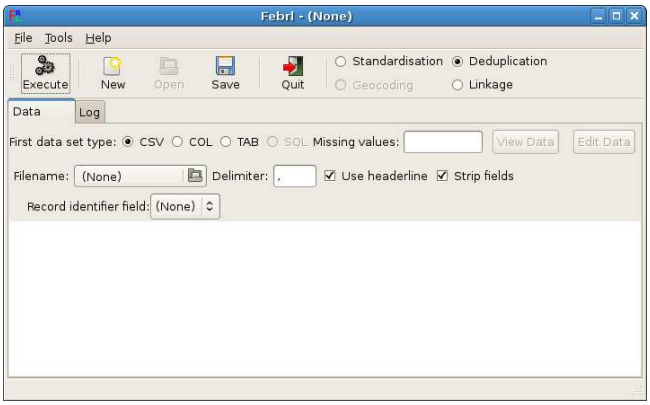

Figure 4: Initial *Febrl* user interface after start-up.

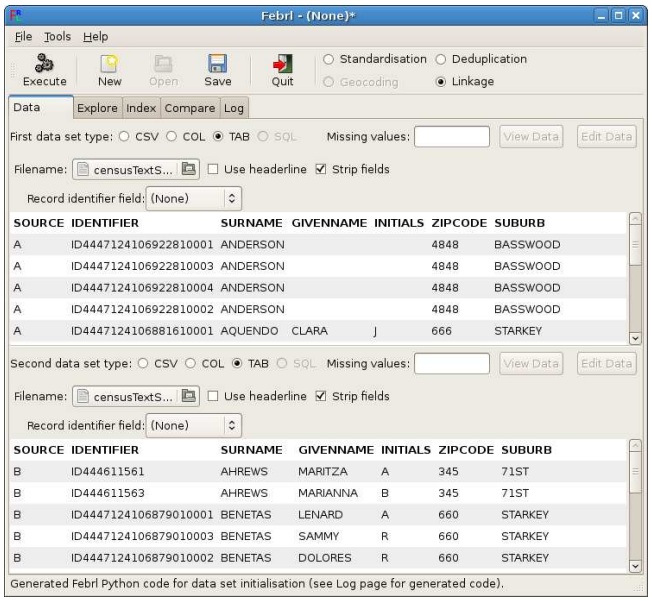

Figure 5: 'Data' tab for a record linkage project after both 'Census' input data sets have been initialised and validated.

as well as for both new and experienced users to learn about, and experiment with such techniques. The current *Febrl* version includes the source code, several example data sets (some taken from the *SecondString* toolkit[3]), and a data set generator [11]. The documentation contains a set of papers that describe the techniques implemented in *Febrl*, and a manual that includes several step-by-step tutorials.

4. STRUCTURE AND FUNCTIONALITY

Compared to earlier versions, *Febrl* since its 0.4 release includes a graphical user interface (GUI), which facilitates the application of *Febrl* for users who do not have any Python programming experience. Feedback received from users since 2002 indicated that a GUI was one of the most desired features for *Febrl*. The GUI was developed using the PyGTK[4] library and the Glade[5] toolkit.

In the following, the structure of the *Febrl* GUI and its functionality are described in detail, and illustrated using a series of screen-shots that show an example linkage of the 'Census' data set, which was taken from the *SecondString* repository and is provided with the *Febrl* software. This small data set is made of 841 records in total. It is split into two sub-sets with 449 and 392 records, respectively, and contains artificial census data made of name and address fields. Each record includes an identifier field, which has the same value if two or more records from the two data sets refer to the same person. Record pairs that have the same identifier value therefore correspond to true matches, while pairs that have different identifier values are non-matches. This allows measuring the accuracy of a linkage [10].

The main *Febrl* GUI after start-up is shown in Figure 4. The basic idea of the GUI is to have a 'tabbed' window, similar to tabs in modern Web browsers. This follows the approach taken by the *Rattle* open source data mining tool [22]. There is one tab for each of the major steps in the record linkage process. On each tab, the user can set a variety of parameter settings for the corresponding step of the linkage process. A click on the 'Execute' button will validate the chosen settings, and if any of them are invalid an error window will appear that describes which setting has to be corrected. Once all settings on a tab are valid, *Febrl* Python code for the corresponding step in the linkage process will be generated and shown in the 'Log' tab. This is the code that will be executed when the actual standardisation, deduplication or linkage is being started on the 'Output/Run' tab, as will be described in Section 4.7.

Initially, the only two tabs visible are 'Data' and 'Log'. Other tabs will appear once the input data set has, or sets have, been initialised. In the middle top part of the *Febrl* GUI, the user can select the type of project she or he wants to conduct: the 'Standardisation' of one data set, the 'Deduplication' of one data set, or the 'Linkage' of two data sets. All tabs will be described in more detail and illustrated with corresponding screen-shots in the following sections.

4.1 Input Data Initialisation

When a user selects the type of project to be conducted, the 'Data' tab will show either one or two input data set areas. The user now needs to select the file name(s) of the input data set(s) to be used. So far, *Febrl* supports three types of text file formats: CSV (comma separated values), TAB (tabulator separated values), and COL (column oriented values with fixed-width fields). Access to databases is of the many features left for future work.

Once a file has been selected, the content of its first few lines will be shown. This allows a user to check the selected parameter settings (such as using a header line, stripping whitespaces from values, setting missing values, and using a record identifier field), or modify them if required. When satisfied, a click on 'Execute' will validate the chosen settings, and provide an error window if any is not valid.

As can be seen in the top part of Figure 5 for the 'Census' data sets, once the input data has been initialised and validated, additional tabs will become visible. The 'Explore' tab will become visible in any case, other visible tabs however depend upon the selected project type.

[3] `http://secondstring.sourceforge.net`
[4] `http://www.pygtk.org`
[5] `http://glade.gnome.org`

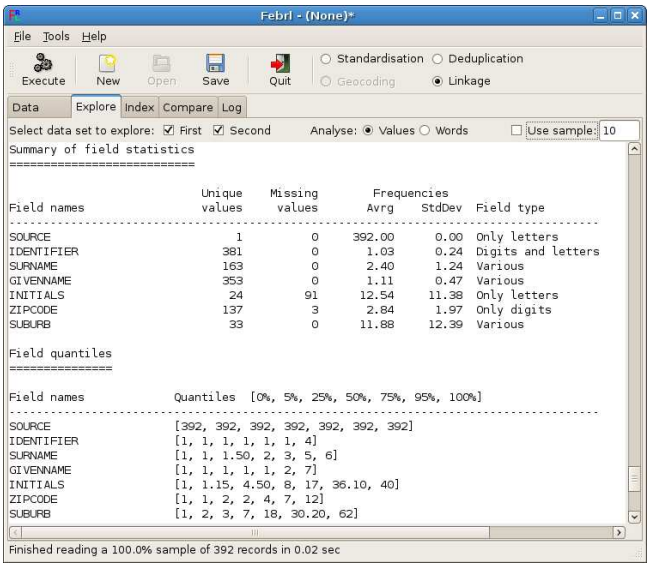

Figure 6: Data exploration tab showing summary analysis of record fields (attributes, columns).

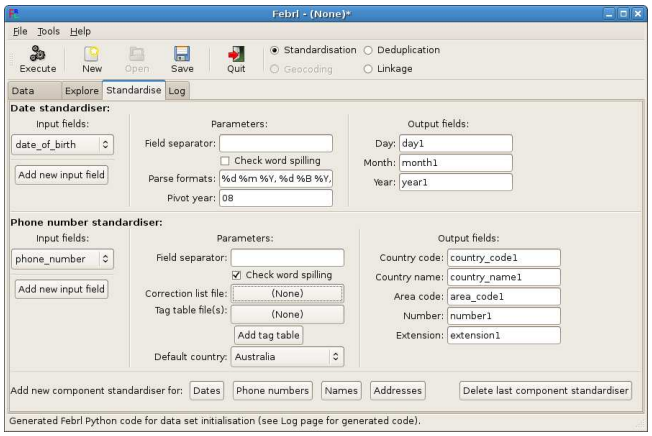

Figure 7: Example date and telephone number standardisers (for a synthetic *Febrl* data set).

4.2 Data Exploration

The 'Explore' tab allows a user to conduct basic data exploration of the input data set(s) in order to get a better understanding of the content of the selected data. The input files will be read and a variety of statistics collected for each field (attribute). This includes the number of different values, the alphabetically smallest and largest values, the most and least frequent values, the quantiles distribution of field values, the number of records that have missing (or empty) values in each field, as well as a guess of the type of each field (if it contains only digits, only letters, or is of mixed type). It is possible to sample a percentage of records to be analysed in order to speed-up the exploration of large data sets. Figure 6 shows a summary table of the information collected from one of the 'Census' data sets.

4.3 Data Cleaning and Standardisation

Currently, the cleaning and standardisation of a data set using the *Febrl* GUI is done separately from a record linkage or deduplication project, rather than as a first step in the linkage process (as shown in Figure 1). A user can clean and standardise her or his data set(s), and they are then written into new file(s), which in turn can be used in a deduplication or record linkage project. When a user selects the 'Standardisation' project type, and has initialised a data set on the 'Data' page, she or he can define one or more component standardisers on the 'Standardise' page, as shown in Figure 7. Currently, *Febrl* contains standardisers for names, addresses, dates, and telephone numbers.

The standardisation for simple names (those made of one given-name and one surname only) is done by applying a rule-based approach, while for more complex names a probabilistic hidden Markov model (HMM) based approach [12] is used. The standardisation of addresses is fully based on a HMM approach [8]. The training of HMMs at the moment needs to be done outside of the *Febrl* GUI using separate *Febrl* programs. The standardisation of dates, such as dates of birth, is based on a list of parsing format strings that provide the most likely date formats that are expected in a record field. Telephone numbers are standardised using an approach that combines look-up tables and rules.

As can be seen in Figure 7, each standardiser requires the user to select one or several input fields from the input data set (shown on the left side in the GUI), which are to be cleaned and standardised into a number of output fields (six for names, 27 for addresses, three for dates, and five for phone numbers), as shown on the right side in the GUI. Each component standardiser also requires various parameters to be set, as shown in the middle column of the GUI.

It is possible to define more than one standardiser for each component type. For example, for a health data set, one date standardiser might be used for dates of birth and another for hospital admission dates. Once all parameter settings for the defined component standardisers are initialised, they can be validated with a click on 'Execute'. On the 'Output/Run' tab (which will be described in Section 4.7), the name of the standardised output file needs to be provided, and then a standardisation project can be started by clicking on the 'Execute' button.

4.4 Indexing (Blocking) Definition

Blocking or indexing is applied in the record linkage process to reduce the number of record pair comparisons to be conducted [2]. The 'Index' tab allows the user to select one of seven possible indexing methods, as shown in Figure 9 (a). The most simple approach is the 'FullIndex', which will compare all record pairs and thus has a quadratic complexity (making it not scalable!). The standard 'BlockingIndex' [2] approach, as implemented in most traditional record linkage systems, inserts each record into a single block, and only compares records within the same block.

Febrl also contains five recently developed experimental indexing methods: 'SortingIndex', which is based on the sorted neighbourhood approach [20]; 'QGramIndex', which uses sub-strings of length q to improve approximate matching [2]; 'CanopyIndex', which is based on overlapping canopy clustering and uses the TF-IDF or Jaccard similarity to cheaply calculate the similarities between records [15]; 'StringMapIndex', which maps the index key values (more on this below)

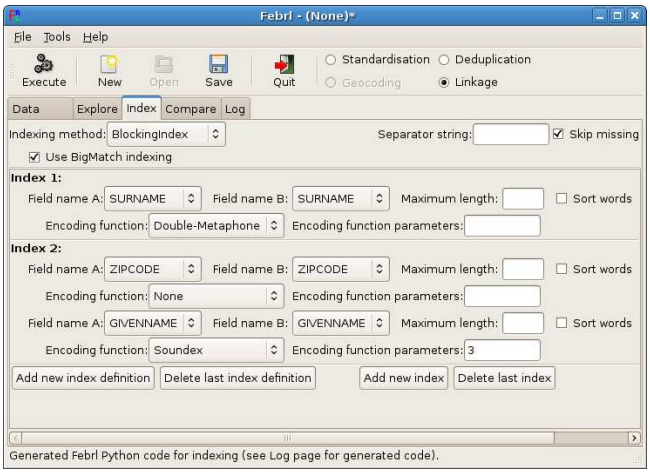

Figure 8: Example indexing definition using the 'BlockingIndex' method and two index definitions.

into a multi-dimensional space and performs canopy clustering on the objects in this space [21]; and 'SuffixArrayIndex', which generates suffix strings of the index key values and inserts them into a sorted array, with the aim to enable efficient access to these values during the record pair comparison step [1].

When conducting a linkage and using one of the indexing methods 'BlockingIndex', 'SortingIndex' or 'QGramIndex', the *BigMatch* [24] approach can be activated in the GUI (as can be seen in Figure 8). With this approach, the smaller of the two input data sets is loaded and an index data structure is built in main memory. It will include all record fields required in the comparison step. Each record of the larger input data set is then read, its index key values are extracted, all records in the same block from the smaller data set are retrieved from the main memory index, and they are compared with the current record from the larger data set. This approach performs only a single pass over the large data set and does not require indexing, sorting or storing of any of its records. This can significantly reduce the run time if two data sets of different sizes are to be linked.

Similarly, for the deduplication of a single data set, the indexing step can be performed in an overlapping fashion with the field comparison step. An inverted index data structure is built in memory while records are read from the input data set, and their index key values are extracted and inserted into this index. At the same time, the current record is compared with all previously read and indexed records that have the same index key value. This approach can be selected by ticking the 'Use Dedup indexing' box.

Once an indexing method has been selected, the actual index keys have to be defined and their various parameters have to be set. Index keys consist of one field (attribute) or a concatenation of several fields. Phonetic encoding functions can be used to group similar sounding values into the same block. The encoding functions implemented in *Febrl* [3] are listed in Figure 9 (b).

Figure 8 shows an example *Febrl* 'Index' tab for the 'Census' data sets. As can be seen, a user has selected the standard 'BlockingIndex' method and has defined two index keys.

Figure 9: Available methods for indexing (a), phonetic encoding (b), field comparison (c), and weight vector classification (d). These are the pull-down menus from the corresponding *Febrl* GUI tabs.

The first will generate key values from the 'SURNAME' field encoded with the Double-Metaphone algorithm [3]. The second index key will be generated by concatenating values from the 'ZIPCODE' field with the first three characters of the Soundex [3] encoded 'GIVENNAME' field. Records that have the same value in either of these two index key definitions will be inserted into the same block and compared in the record pair comparison step.

4.5 Field Comparison Functions

On the 'Comparison' tab, the functions used to compare the field values of record pairs can be selected. *Febrl* contains 26 similarity functions, including twenty approximate string comparators [3], as listed in Figure 9 (c). Every comparison requires the user to select a comparison function, as well as the two fields that will be used in this comparison. While normally fields with the same content will be compared (for example surnames with surnames), it is feasible in *Febrl* to compare different fields, for example to accommodate for swapped given- and surname values. While most of the comparison functions implemented in *Febrl* are variations of approximate string comparisons [3; 14], special functions are available that allow the user to compare fields that contain date, age, time or numerical values.

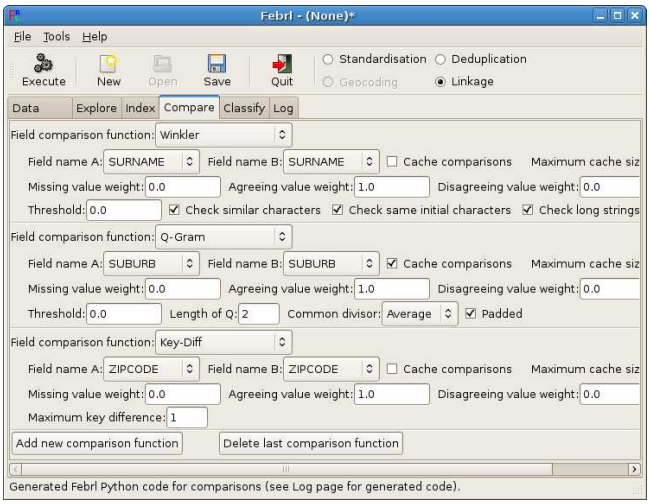

Figure 10: An example of three field comparison function definitions.

All comparison functions return a raw similarity value between 0 (total dissimilarity) and 1 (exact match). There is no limit to the number of comparison functions that can be initialised. Because similarity functions can be computationally quite expensive, especially when longer strings are compared, it is possible to cache the compared field values together with their similarity value. This will speed-up the comparison step for all subsequent comparisons of the same two field values. Caching is especially useful for fields that contain a small number of longer string values, such as suburb, business or company names, or article titles.

The example comparison functions shown in Figure 10 apply the Winkler [3] approximate string comparator on the 'SURNAME' fields, a q-gram based approximate string comparator on the 'SUBURB' fields, and the key-difference comparison function (which counts the number of different characters) on the 'ZIPCODE' fields. For each compared record pair, a weight vector with three matching weights will be generated to be used to classify that pair.

4.6 Weight Vector Classification

The last major step that needs to be initialised (before a linkage or deduplication can be started) is the selection of the method used to classify the weight vectors generated in the comparison step. *Febrl* currently offers six classification techniques, as listed in Figure 9 (d).

With the 'FellegiSunter' classifier, all matching weights in a weight vector are summed, and two manually set thresholds are used to classify record pairs [17]. Those pairs that have a summed weight above the upper threshold are classified as matches, pairs with a matching weight below the lower threshold as non-matches, and pairs with a matching weight between the two thresholds as possible matches.

The 'OptimalThreshold' classifier requires the true match status of all compared record pairs to be known (i.e. it is a supervised classifier). Then, an optimal threshold can be calculated based on the summed weight vectors. Another supervised classifier is 'SuppVecMachine', which implements a support vector machine (SVM). Several SVM parameters,

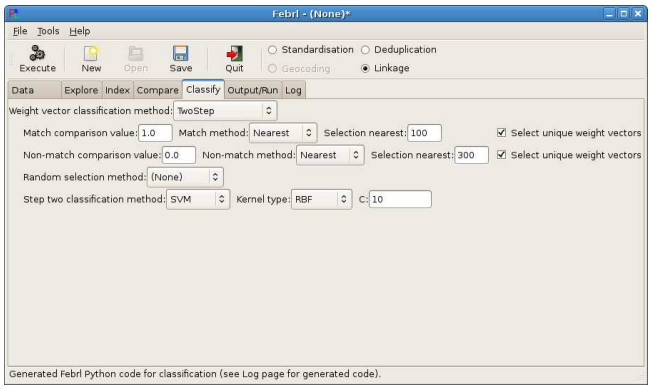

Figure 11: Example 'Two-Step' weight vector classifier.

including the kernel function used, can be set in the GUI. For both supervised classifiers, the match status of record pairs is determined through an exact comparison of one of the fields in the data set(s). For example, the 'Census' data sets contain the 'IDENTIFIER' field. For two records that refer to the same person, an exact comparison on this field will result in a similarity value of 1, because they will have the same identifier value. On the other hand, all comparisons between different people (that have different identifier values) will result in a similarity value of 0. These similarity values can then be used as class indicator variable, which allows supervised classification.

The 'FarthestFirst' [18] and 'KMeans' classifiers are both unsupervised clustering approaches. They cluster weight vectors into matches and non-matches. Several methods can be selected for centroid initialisation, and different distance measures are available. It is possible to sample the weight vectors in order to reduce the computational requirements of the clustering process. Both classifiers also allow the selection of a 'fuzzy region', which will classify the record pairs in the area half-way between the match and non-match centroids as possible matches, as described in [19].

Finally, the unsupervised 'TwoStep' classifier, shown in Figure 11, is based on the idea of selecting in a first step weight vectors that with high likelihood correspond to true matches and true non-matches, and to use these vectors in a second step as training examples for a binary classifier [4; 5; 6]. Several methods are implemented in *Febrl* on how to select the training examples in the first step, and for the second step k-means clustering or a SVM classifier can be selected. Experiments have shown that in certain cases this unsupervised two-step approach can achieve linkage quality results almost as good as supervised classification [4].

4.7 Output Files and Running a Project

On the 'Output/Run' tab, a user can select various settings of how the weight vectors, the match status, the matched record pairs, and the matched files can be saved into output files. It also allows setting several other parameters related to running a project, such as setting a length filter (i.e. removing candidate record pairs if their lengths, as concatenated strings, differs by at least a certain percentage value), or setting a cut-off threshold to reduce the number of weight vectors that are stored in memory.

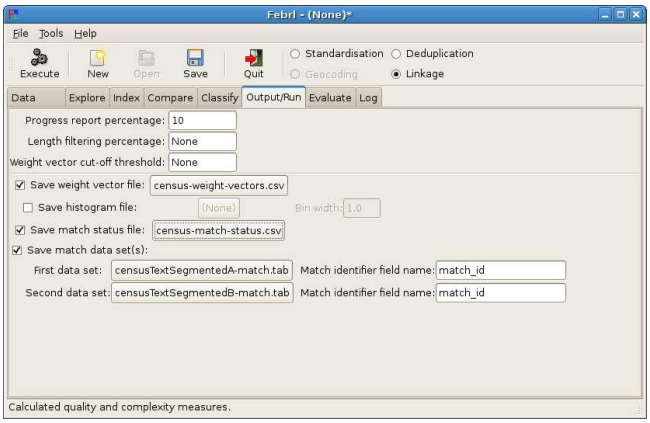

Figure 12: 'Output/Run' tab with options for running a linkage project and writing of output files.

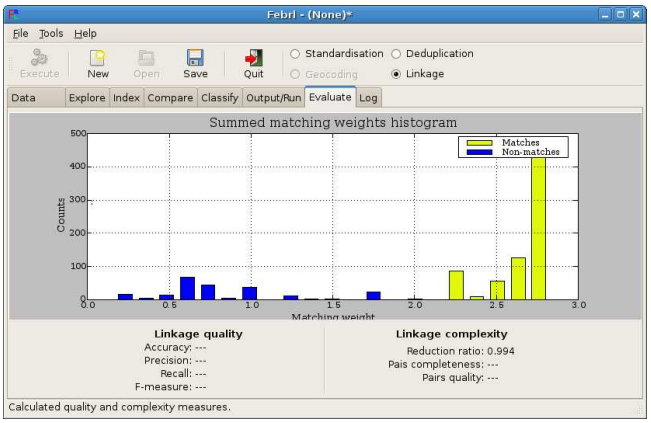

Figure 13: Evaluation tab showing the matching weight histogram and quality and complexity measures for a linkage.

Figure 12 shows an example 'Output/Run' tab for the 'Census' linkage project. With a click on 'Execute', the *Febrl* GUI will ask the user if the current project should be saved as a *Febrl* Python file, that can later be execute independently from the GUI; and if the project should be run within the GUI now. Once started, a small window will appear that shows a progress bar as the project is being run.

4.8 Evaluation and Clerical Review

As can be seen in Figure 13, the 'Evaluate' page visualises the results of a deduplication or linkage as a histogram of the summed matching weights of the compared record pairs. If the true match and non-match status of record pairs is available (as discussed in Section 4.6), the quality of the conducted linkage or deduplication will be shown using the measurements accuracy, precision, recall and F-measure (or F-score) [10]. Additional measures that show the complexity of a deduplication or linkage project are the reduction ratio, pairs completeness and pairs quality [10]. They are based on the number of compared record pairs, the total number of possible pairs (i.e. if each record would have been compared with all others), and if these pairs are true matches or not.

4.9 Log Tab

This tab shows the *Febrl* Python code generated when clicking 'Execute' on other GUI tabs. An example is shown in Figure 14, where the code generated for a k-means record pair classifier can be seen. This allows an experienced *Febrl* user to verify the correctness of the generated code, and also enables copying pieces of code into other *Febrl* Python programs outside of the GUI.

5. USER EXPERIENCES

In order to get a feel of how and by whom *Febrl* is being used, in early 2009 the author sent out an e-mail questionnaire to forty *Febrl* users that are currently registered on the *Febrl Sourceforge.Net* mailing list, and to an additional 37 users who had e-mail contact with the author in the past two years. Of these 77 e-mails, eight were bounced with an error message, mostly because an e-mail did not exist anymore. Of the 69 e-mails sent successfully, 22 users responded to the questionnaire, corresponding to a 32% response rate.

Of the respondents, 27% indicated they worked in industry, 18% in government, and the remaining 55% in academia. A large variety of application areas, were the respondents worked in, was given. Most common were the health sector (four respondents), computer science research and the census (three respondents each), and business (two respondents). Other areas included demographics, security, telecommunication, data warehousing and social research.

Table 2: Years when users became aware of *Febrl*.

2002	2003	2004	2005	2006	2007	2008
4.6%	13.7%	9%	4.6%	9%	18.2%	40.9%

The year when the respondents became aware of *Febrl* is shown in Table 2. Fifteen respondents (77%) indicated that they have been using *Febrl* since they became aware of it, however only nine (40.9%) indicated that they are currently using it. The roles in which *Febrl* is being used is listed in Table 3. Note that a respondent could select several of these roles. As can be seen, learning about record linkage and experimental linkages were the two most common roles *Febrl* is being used for. Interestingly, more than a quarter of all respondents replied that they are, or were, using *Febrl* within production linkage projects. Exactly half of the respondents also indicated that they only used some of *Febrl*'s functionality (such as its string comparison or phonetic encoding modules, or the *Febrl* data generator), for example in their research.

Table 3: Role in which *Febrl* is being used.

Just playing around with it	54.5%
Experimental linkages	59.1%
Production linkages	27.3%
Data cleaning only	18.2%
Learning about record linkage	68.2%
Record linkage research	40.9%

Nearly 70% of the respondents (15 of 22) were using or exploring other record linkage systems besides *Febrl*. These included other open source systems (*Kettle* by Penthao, the *Link King*, and Sun's *Mural*), various commercial products,

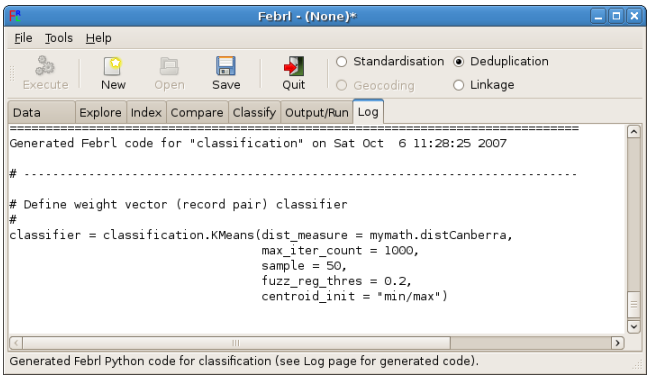

Figure 14: Log tab showing the *Febrl* Python code generated for an example 'KMeans' classifier.

as well as specific solutions developed in-house. More than three-quarters of the respondents (77%) affirmed that their choice of *Febrl* was influenced by it being open source software. The justifications for this answer ranged from strong believers in the open source philosophy, to the more pragmatic views of no costs involved, or that it was important to have access to the source code in order to be able to compare record linkage algorithms for research.

One of the biggest improvements in *Febrl* version 0.4 was the addition of a GUI. Surprisingly, however, only a bit more than half (53%) of all respondents replied that the inclusion of the GUI made a big difference in their appreciation of *Febrl*. This can partially be explained by the fact that nearly half of all respondents have used *Febrl* before the GUI was released (November 2007), and they were therefore used to configure it by modifying or writing Python programs.

As *Febrl*'s advantages, respondents listed: being well documented and its references to published research (*Febrl* is not based on trade secrets); being highly configurable and extendible; the availability of its source code; the variety of techniques implemented for all steps of the record linkage process; and the inclusion of a data generator.

On the other hand, according to the respondents, *Febrl*'s major disadvantages include its poor scalability; its requirement of large amounts of memory for large data sets; its slowness (because it is implemented in Python); missing handling of linked data (merging of linked records); error messages that are not always clear; the small community which means help is not easy to get; a complex installation procedure (no one-in-all installer available); the requirement of Python skills to configure *Febrl*; no direct database access; and only limited support from the developers.

Overall, most respondents were pleased about this freely available data cleaning, deduplication and record linkage system. *Febrl* gave them the opportunities to learn more about the techniques used for these tasks, and allowed them to conduct practical experimental linkages, something that would not have been possibly without *Febrl*.

6. CONCLUSIONS AND FUTURE WORK

From a developer's point of view, the *Febrl* project has been – and still is – an interesting experience. On one hand, the development of software that can be published requires a much increased effort compared to writing research prototype software that is only used for experimental evaluations. The development of the *Febrl* GUI especially was a very time consuming effort. On the other hand, the feedback received from users, and the contacts gained with record linkage researchers and practitioners worldwide, would not have been possible without *Febrl*.

The *Febrl* software has made an impact in the areas of data cleaning, deduplication and record linkage. However, measuring the impact of such an open source project is not simple. Download numbers on one hand, and the number of users registered on a mailing list on the other hand, seem to be the two ends of the impact spectrum (one might be too high, the other too low). For application oriented software such as *Febrl*, which is primarily used by people other than computer science researchers, the feedback received from users can be very limited.

If open source software like *Febrl* is used within organisations for experimental or even for production linkages, then commonly this is not acknowledged by the organisation in reports that present results of such linkage projects, or on their corresponding Web sites. This can be quite frustrating from the point of view of an academic who needs to be able to prove the impact of her or his research (besides academic citation numbers), in order to successfully progress in her or his academic career.

Ease of installation on all popular operating systems is likely to be a major critical factor that can make or break the success of an open source application software. If potential users cannot install open source software in the same way as commercial software, they will likely become easily frustrated and abandon the installation process (*Febrl* currently requires manual installation of various Python modules). Users outside of the computer science and information technology domains, for example people working in the health sector or the social sciences, might not have the skills required for a complicated installation process, and therefore might quickly give up on using software that does not provides a simple and automatic installation procedure. Requests to support and help with installation on different systems can be outside of the expertise of the developers.

Besides reports on bugs in the *Febrl* software, a common topic of feedback by users to the *Febrl* developers is the question of when and if certain features will be added to the software. While most of such features would be of general interest and some are already in *Febrl*'s to-do list (such as completing the GUI functionalities or adding clerical review support), other requests are specific to a certain domain (e.g. the health sector), country (like providing country specific look-up files for data cleaning), or application area (for example special field comparison functions for business related data). Due to the limited resources (mainly time) of the *Febrl* developers, it is unlikely that such features will ever be added, unless significant resources will become available to the developers, or parts of the development are taken over by other individuals or organisations.

As mentioned in the previous paragraph, there is a long (wish) list of features that should be added to the *Febrl* software at some stage. Completing the functionality of the GUI, and including the data generator and the HMM training modules into the GUI are three major pieces of development that are required. Adding further quality measures, such as ROC curve or AUC, and an interactive feature

that allows manipulation of a classification threshold on the histogram shown on the 'Evaluate' tab, would be a major benefit. It would allow users to play with the classification threshold and immediately see the resulting changes in linkage quality measures. Another major addition to the GUI would be a 'Review' tab which allows users to view and manually classify record pairs as matches or non-matches that were originally classified as possibly matches. The manually classified record pairs can then be used as training examples, by feeding their match status back to the classifier (as shown in Figure 1), facilitating the improvement of the deduplication or linkage quality.

Apart from the GUI, additional output options should be added to *Febrl* that allow flexible merging of the linked records into a linked output data set. Providing access to SQL and ODBC databases, in order to load input data from a database and write the linked output data back into a database, would allow the integration of *Febrl* into a variety of database environments. Implementing additional methods for field comparison, classification and indexing would extend *Febrl*'s utility as an experimental platform.

Another avenue of work that would make *Febrl* more versatile and applicable for practical use will be to improve the performance of the core modules, and at the same time reducing the amount of memory required when deduplicating or linking larger data sets. Performance can be increased by replacing the core comparison functions and indexing data structures, currently written in Python, with corresponding C code. Given the increasing availability of multi-core parallel computing platforms, an orthogonal way of increasing performance will be to develop parallel versions of all core *Febrl* modules (note that version 0.3 of *Febrl* did include some experimental parallelisation approaches [9]).

7. ACKNOWLEDGEMENTS

This research has been supported by the Australian Research Council (ARC) under Linkage Project LP0453463. It was partially funded by the New South Wales Department of Health, Sydney, with additional funding provided by the Australian Partnership for Advanced Computing. The author would like to thank everybody who has contributed to the *Febrl* project over the years, and also thank all *Febrl* users who responded to the questionnaire.

8. REFERENCES

[1] A. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *International Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05)*, pages 30–39, Tokyo, 2005.

[2] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD'03 workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 25–27, Washington DC, 2003.

[3] P. Christen. A comparison of personal name matching: Techniques and practical issues. In *Workshop on Mining Complex Data (MCD'06), held at IEEE ICDM'06*, Hong Kong, 2006.

[4] P. Christen. A two-step classification approach to unsupervised record linkage. In *Australasian Data Mining Conference (AusDM'07), Conferences in Research and Practice in Information Technology (CRPIT)*, volume 70, pages 111–119, Gold Coast, Australia, 2007.

[5] P. Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'08)*, pages 151–159, Las Vegas, 2008.

[6] P. Christen. Automatic training example selection for scalable unsupervised record linkage. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'08), Springer LNAI 5012*, pages 511–518, Osaka, Japan, 2008.

[7] P. Christen. Febrl – An open source data cleaning, deduplication and record linkage system with a graphical user interface (Demonstration Session). In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'08)*, pages 1065–1068, Las Vegas, 2008.

[8] P. Christen and D. Belacic. Automated probabilistic address standardisation and verification. In *Australasian Data Mining Conference (AusDM'05)*, pages 53–67, Sydney, 2005.

[9] P. Christen, T. Churches, and M. Hegland. Febrl – A parallel open source data linkage system. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04), Springer LNAI 3056*, pages 638–647, Sydney, 2004.

[10] P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In F. Guillet and H. Hamilton, editors, *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 127–151. Springer, 2007.

[11] P. Christen and A. Pudjijono. Accurate synthetic generation of realistic personal information. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'09), Springer LNAI 5476*, pages 507–514, Bangkok, Thailand, 2009.

[12] 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*, 2(9), 2002.

[13] D. E. Clark. Practical introduction to record linkage for injury research. *British Medical Journal*, 10(3):186–191, 2004.

[14] W. W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Workshop on Information Integration on the Web (IIWeb'03), held at IJCAI'03*, pages 73–78, Acapulco, 2003.

[15] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*, pages 475–480, Edmonton, 2002.

[16] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[17] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64(328):1183–1210, 1969.

[18] K. Goiser and P. Christen. Towards automated record linkage. In *Australasian Data Mining Conference (AusDM'06), Conferences in Research and Practice in Information Technology (CRPIT)*, volume 61, pages 23–31, Sydney, 2006.

[19] L. Gu and R. Baxter. Decision models for record linkage. In *Selected Papers from AusDM, Springer LNCS 3755*, pages 146–160, 2006.

[20] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *ACM International Conference on Management of Data (SIGMOD'95)*, pages 127–138, San Jose, 1995.

[21] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications (DASFAA'03)*, pages 137–146, Tokyo, 2003.

[22] G. Williams. Data mining with Rattle and R. Togaware, Canberra, 2009. Software available at: `http://rattle.togaware.com`.

[23] W. Winkler. Methods for evaluating and creating data quality. *Elsevier Information Systems*, 29(7):531–550, 2004.

[24] W. E. Yancey. BigMatch: A program for extracting probable matches from a large file for record linkage. Technical Report RR2007/01, US Bureau of the Census, 2007.