Febrl – An Open Source Data Cleaning, Deduplication and Record Linkage System with a Graphical User Interface

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

ABSTRACT

Matching records that refer to the same entity across databases is becoming an increasingly important part of many data mining projects, as often data from multiple sources needs to be matched in order to enrich data or improve its quality. Significant advances in record linkage techniques have been made in recent years. However, many new techniques are either implemented in research proof-of-concept systems only, or they are hidden within expensive 'black box' commercial software. This makes it difficult for both researchers and practitioners to experiment with new record linkage techniques, and to compare existing techniques with new ones. The *Febrl* (Freely Extensible Biomedical Record Linkage) system aims to fill this gap. It contains many recently developed techniques for data cleaning, deduplication and record linkage, and encapsulates them into a graphical user interface (GUI). *Febrl* thus allows even inexperienced users to learn and experiment with both traditional and new record linkage techniques. Because *Febrl* is written in Python and its source code is available, it is fairly easy to integrate new record linkage techniques into it. Therefore, *Febrl* can be seen as a tool that allows researchers to compare various existing record linkage techniques with their own ones, enabling the record linkage research community to conduct their work more efficiently. Additionally, *Febrl* is suitable as a training tool for new record linkage users, and it can also be used for practical linkage projects with data sets that contain up to several hundred thousand records.

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

General Terms: Algorithms, Experimentation

Keywords: Data matching, data linkage, deduplication, data cleaning, open source software, Python

1. PROJECT BACKGROUND

Febrl has been developed since 2002 as part of a collaborative research project conduced between the Australian National University in Canberra and the New South Wales Department of Health in Sydney, Australia. The objective of this project is to develop new techniques for improved data cleaning and standardisation, deduplication and record linkage within the domain of health databases.

The *Febrl* system is written in the programming language Python[1], which is an ideal platform for rapid prototype development, as it provides data structures such as sets, lists and dictionaries (associative arrays) that allow efficient handling of very large data sets. It also includes many modules offering a large variety of functionalities. It has excellent built-in string handling capabilities, and the large number of extension modules facilitate, for example, database access and GUI development. The *Febrl* GUI is based on the PyGTK[2] library and the Glade[3] toolkit.

Febrl is published under an open source software licence. Due to the availability of its source code, *Febrl* is suitable for the rapid development, implementation, and testing of novel record linkage algorithms and techniques, as well as for both new and experienced users to learn about, and experiment with, various record linkage techniques. Since 2002 *Febrl* has been hosted on the *Sourceforge.Net* repository at:

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

The total number of downloads of *Febrl* files on 14th June 2008 has reached 9,840. To the best of the author's knowledge, *Febrl* is the only open source record linkage system with a GUI that allows data cleaning and standardisation, deduplication and record linkage. The current *Febrl-0.4* version includes the source code, several example data sets (as available from the *SecondString* toolkit[4]), and a data set generator. The documentation contains several papers that describe the techniques implemented in *Febrl*, and a manual that includes several step-by-step tutorials.

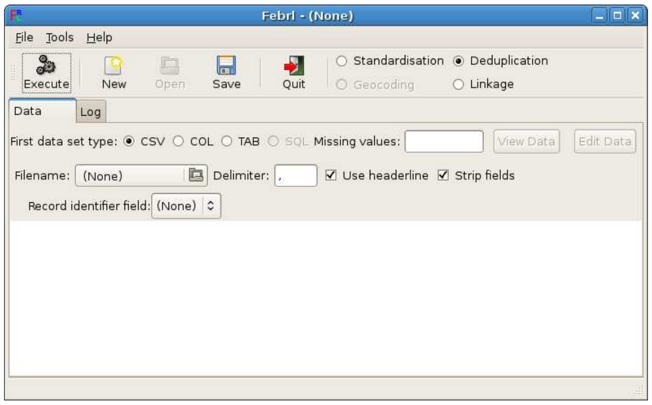
Compared to earlier versions, *Febrl-0.4* not only contains a GUI, but also a variety of new techniques, as described below. Some of the screenshots following show an example linkage of the 'Census' data set from *SecondString*.

2. STRUCTURE AND FUNCTIONALITY

The *Febrl* GUI has been developed with the objective to make *Febrl* more accessible to non-technical record linkage users [8]. The structure of the *Febrl* GUI follows the *Rattle* open source data mining tool [17]. The basic idea is to have a window that contains one tab (similar to tabs in modern Web browsers) per major step of the record linkage process. The start-up *Febrl* GUI is shown in Figure 1. First, only two tabs are visibly, additional tabs will appear once the input data has been initialised.

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

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.
ACM 978-1-60558-193-4/08/08.



Standardisation [✓ Deduplication] Geocoding Linkage

First data set type: [✓ CSV] COL TAB SQL Missing values:

Filename: (None) Delimiter: , [✓ Use headerline] [✓ Strip fields]

Record identifier field: (None)

Figure 1: Initial *Febrl* user interface after start-up.

On each tab, the user can select methods and their parameters, and then confirm these settings by clicking the 'Execute' button. Corresponding *Febrl* Python code will be generated and shown in the 'Log' tab. Once all necessary steps are set up, the generated Python code can be saved and run outside of the GUI, or a project can be started and its results can be evaluated from within the *Febrl* GUI. All major tabs will be described in more detail and illustrated with corresponding screenshots in the following sections.

2.1 Input Data Initialisation

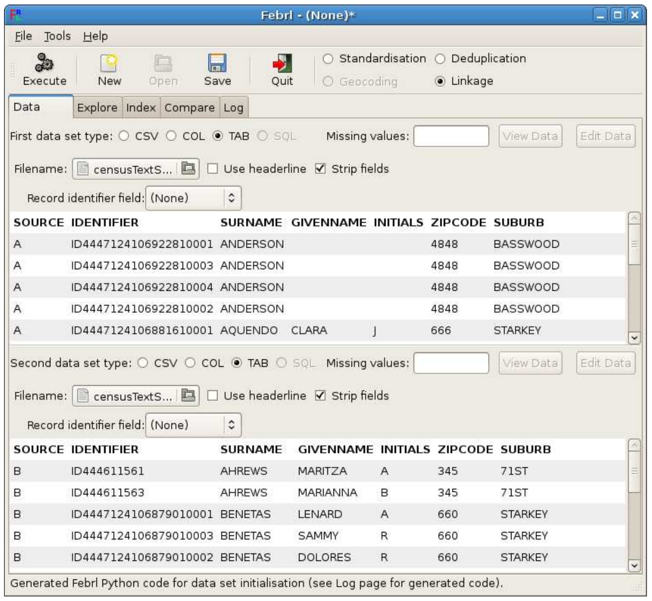
A user first has to select the type of project to be conducted: (a) cleaning and standardisation of a data set, (b) deduplication of a data set, or (c) linkage of two data sets. The 'Data' tab of the *Febrl* GUI will change accordingly and either show one or two input data set selection areas. Currently, several text based file formats are supported, including the commonly used CSV (comma separated values) format. Once a file has been selected, its first few lines will be shown, as illustrated in Figure 2. This enables the user to verify the chosen settings, or adjust them if required. When satisfied, a click on the 'Execute' button will confirm the settings, and the tabs for data exploration and, depending upon the project type selected, standardisation, or indexing, comparison and classification will become visible.

2.2 Data Exploration

The 'Explore' tab enables the user to analyse the input data set(s) to get a better understanding of its/their content and quality. After a click on the 'Execute' button, the data set(s) will be read and all fields (or attributes, columns) will be analysed. A report will be displayed that for each field provides information about the number of different values in it, the alphabetically smallest and largest values, the most and least frequent values, the quantiles distribution of the values, the number of records with missing values, as well as a guess of the type of the field (if it contains only digits, only letters, or is of mixed type). A summary table of this analysis is then displayed, as shown in Figure 3.

2.3 Data Cleaning and Standardisation

Data cleaning and standardisation using the *Febrl* GUI is currently done separately from a linkage or deduplication. A data set can be cleaned and standardised, and is



Standardisation Deduplication Geocoding [✓ Linkage]

First data set type: CSV COL [✓ TAB] SQL Missing values:

Filename: censusTextS... Use headerline [✓ Strip fields]

Record identifier field: (None)

SOURCE	IDENTIFIER	SURNAME	GIVENNAME	INITIALS	ZIPCODE	SUBURB
A	ID4447124106922810001	ANDERSON			4848	BASSWOOD
A	ID4447124106922810003	ANDERSON			4848	BASSWOOD
A	ID4447124106922810004	ANDERSON			4848	BASSWOOD
A	ID4447124106922810002	ANDERSON			4848	BASSWOOD
A	ID4447124106881610001	AQUENDO	CLARA	J	666	STARKEY

Second data set type: CSV COL [✓ TAB] SQL Missing values:

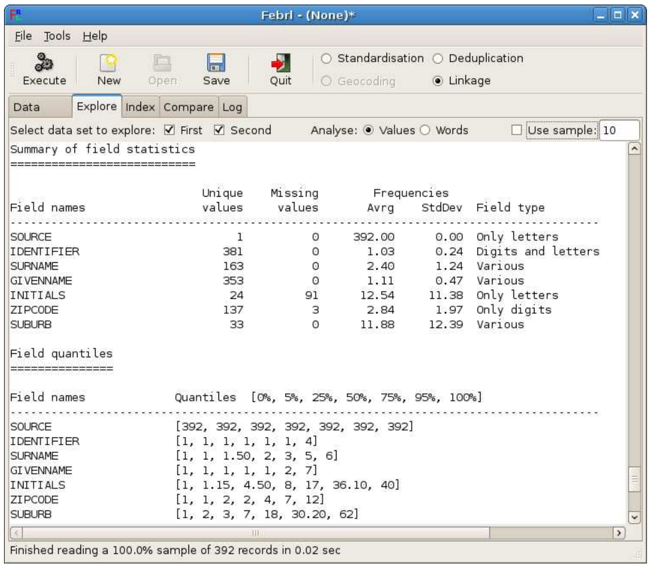
Filename: censusTextS... Use headerline [✓ Strip fields]

Record identifier field: (None)

SOURCE	IDENTIFIER	SURNAME	GIVENNAME	INITIALS	ZIPCODE	SUBURB
B	ID444611561	AHREWS	MARITZA	A	345	71ST
B	ID444611563	AHREWS	MARIANNA	B	345	71ST
B	ID4447124106879010001	BENETAS	LENARD	A	660	STARKEY
B	ID4447124106879010003	BENETAS	SAMMY	R	660	STARKEY
B	ID4447124106879010002	BENETAS	DOLORES	R	660	STARKEY

Generated Febrl Python code for data set initialisation (see Log page for generated code).

Figure 2: *Febrl* user interface for a linkage project after the 'Census' input data sets have been initialised.



Standardisation Deduplication Geocoding [✓ Linkage]

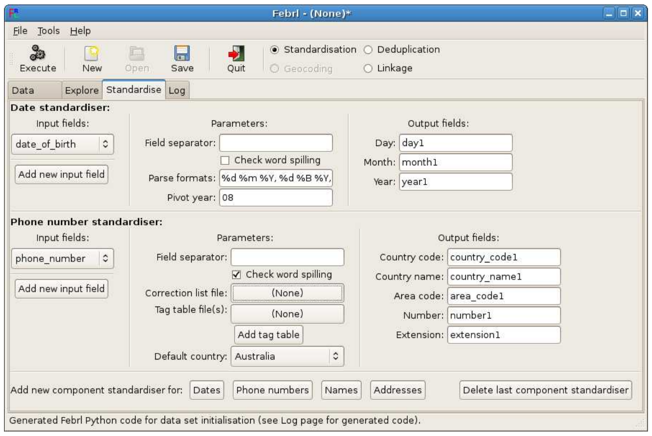
Select data set to explore: [✓ First] [✓ Second] Analyse: [✓ Values] Words Use sample: 10

```
Summary of field statistics
===========================

                            Unique    Missing      Frequencies
Field names                 values     values      Avrg    StdDev  Field type
--------------------------------------------------------------------------------
SOURCE                           1          0    392.00      0.00  Only letters
IDENTIFIER                     381          0      1.03      0.24  Digits and letters
SURNAME                        163          0      2.40      1.24  Various
GIVENNAME                      353          0      1.11      0.47  Various
INITIALS                        24         91     12.54     11.38  Only letters
ZIPCODE                        137          3      2.84      1.97  Only digits
SUBURB                          33          0     11.88     12.39  Various

Field quantiles
===============

Field names                 Quantiles  [0%, 5%, 25%, 50%, 75%, 95%, 100%]
--------------------------------------------------------------------------------
SOURCE                      [392, 392, 392, 392, 392, 392, 392]
IDENTIFIER                  [1, 1, 1, 1, 1, 1, 4]
SURNAME                     [1, 1, 1.50, 2, 3, 5, 6]
GIVENNAME                   [1, 1, 1, 1, 1, 2, 7]
INITIALS                    [1, 1.15, 4.50, 8, 17, 36.10, 40]
ZIPCODE                     [1, 1, 2, 2, 4, 7, 12]
SUBURB                      [1, 2, 3, 7, 18, 30.20, 62]
```

Finished reading a 100.0% sample of 392 records in 0.02 sec

Figure 3: Data exploration tab showing summary analysis of record fields (or attributes, columns).

then written into a new data set, which in turn can then be deduplicated or used for a linkage. Currently, *Febrl* contains standardisers for names, addresses, dates, and telephone numbers. The name standardiser uses a rule-based approach for simple names (such as those made of one given- and one surname only) in combination with a probabilistic hidden Markov model (HMM) approach for more complex names [11], while address standardisation is fully based on a HMM approach [9]. These HMMs currently have to be trained outside of the *Febrl* GUI, using separate *Febrl* modules. Dates are standardised using a list of format strings


Febrl - (None)*
File Tools Help
Execute New Open Save Quit
[✓ Standardisation] Deduplication Geocoding Linkage
Data Explore Standardise Log

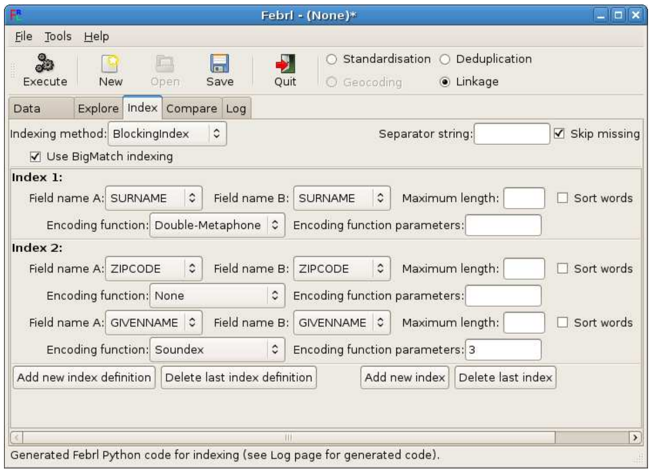
Date standardiser:
Input fields: date_of_birth
Add new input field
Parameters:
Field separator:
☐ Check word spilling
Parse formats: %d %m %Y, %d %B %Y,
Pivot year: 08
Output fields:
Day: day1
Month: month1
Year: year1

Phone number standardiser:
Input fields: phone_number
Add new input field
Parameters:
Field separator:
☑ Check word spilling
Correction list file: (None)
Tag table file(s): (None)
Add tag table
Default country: Australia
Output fields:
Country code: country_code1
Country name: country_name1
Area code: area_code1
Number: number1
Extension: extension1

Add new component standardiser for: Dates Phone numbers Names Addresses Delete last component standardiser

Generated Febrl Python code for data set initialisation (see Log page for generated code).

Figure 4: Example date and telephone number standardisers (for a synthetic *Febrl* data set).


Febrl - (None)*
File Tools Help
Execute New Open Save Quit
Standardisation Deduplication Geocoding [✓ Linkage]
Data Explore Index Compare Log

Indexing method: BlockingIndex
Separator string:
☑ Skip missing
☑ Use BigMatch indexing

Index 1:
Field name A: SURNAME Field name B: SURNAME Maximum length: ☐ Sort words
Encoding function: Double-Metaphone Encoding function parameters:

Index 2:
Field name A: ZIPCODE Field name B: ZIPCODE Maximum length: ☐ Sort words
Encoding function: None Encoding function parameters:
Field name A: GIVENNAME Field name B: GIVENNAME Maximum length: ☐ Sort words
Encoding function: Soundex Encoding function parameters: 3

Add new index definition Delete last index definition Add new index Delete last index

Generated Febrl Python code for indexing (see Log page for generated code).

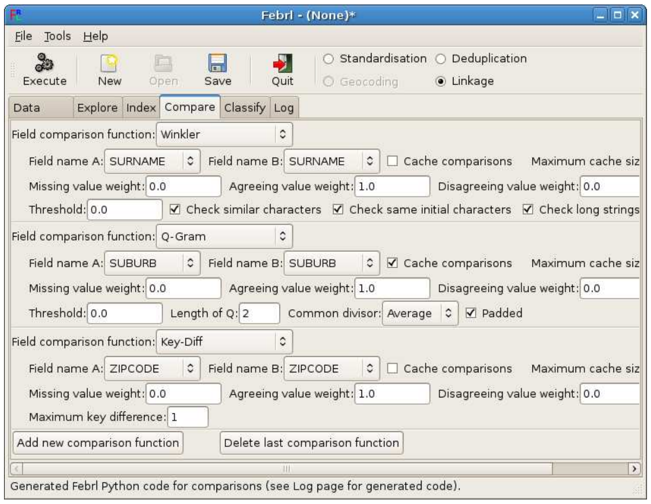
Figure 5: Example indexing definition using the 'BlockingIndex' method and two index definitions.

that provide the expected date formats likely to be found in the input data set. Telephone numbers are also standardised using a rule-based approach. Once initialised and confirmed with a click on 'Execute', on the 'Output/Run' tab (see below) the file name of the standardised output file can be chosen, and a standardisation project can then be started by clicking 'Execute' on the 'Output/Run' tab.

2.4 Indexing (Blocking) Definition

Blocking or indexing is used to reduce the number of detailed record pair comparisons to be done [2]. On the 'Index' tab, a user can select one of seven possible indexing methods. Besides the 'FullIndex' (which will compare all record pairs and thus has a quadratic complexity) and the standard 'BlockingIndex' approach [2] as implemented in many record linkage systems, *Febrl* contains five recently developed indexing methods [4]: 'SortingIndex', which is based on the sorted neighbourhood approach [15]; 'QGramIndex', which uses sub-strings of length q to allow fuzzy blocking [2]; 'CanopyIndex', which employs overlapping canopy clustering using TF-IDF or Jaccard similarity [12]; 'StringMapIndex', which maps the index key values into a multi-dimensional space and performs canopy clustering on these multi-dimensional objects [16]; and 'SuffixArrayIndex', which generates all suffixes of the index key values and inserts them into a sorted array to enable efficient access to the index key values and generation of the corresponding blocks [1].

Once an index method has been chosen, the actual index keys have to be selected and their various parameters have to be set. Index keys are made of one field value, or a concatenation of several field values, that are often phonetically encoded to group similar sounding values into the same block. *Febrl* contains nine encoding methods [3], including Soundex, NYSIIS, Phonix, Phonex, and Double-Metaphone.


Febrl - (None)*
File Tools Help
Execute New Open Save Quit
Standardisation Deduplication Geocoding [✓ Linkage]
Data Explore Index Compare Classify Log

Field comparison function: Winkler
Field name A: SURNAME Field name B: SURNAME ☐ Cache comparisons Maximum cache siz
Missing value weight: 0.0 Agreeing value weight: 1.0 Disagreeing value weight: 0.0
Threshold: 0.0 ☑ Check similar characters ☑ Check same initial characters ☑ Check long strings

Field comparison function: Q-Gram
Field name A: SUBURB Field name B: SUBURB ☑ Cache comparisons Maximum cache siz
Missing value weight: 0.0 Agreeing value weight: 1.0 Disagreeing value weight: 0.0
Threshold: 0.0 Length of Q: 2 Common divisor: Average ☑ Padded

Field comparison function: Key-Diff
Field name A: ZIPCODE Field name B: ZIPCODE ☐ Cache comparisons Maximum cache siz
Missing value weight: 0.0 Agreeing value weight: 1.0 Disagreeing value weight: 0.0
Maximum key difference: 1

Generated Febrl Python code for comparisons (see Log page for generated code).

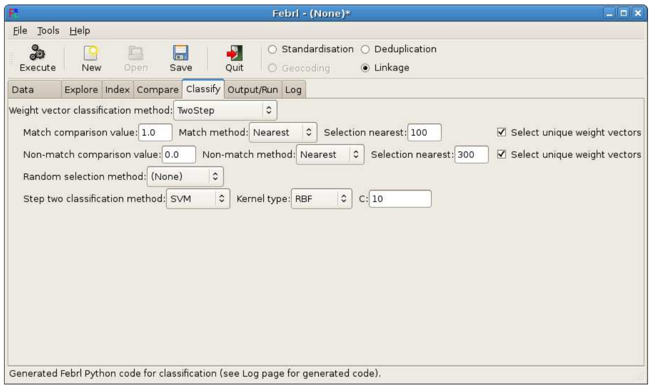
Figure 6: An example of three field comparison function definitions.

2.5 Field Comparison Functions

The similarity functions used to compare the field (attribute) values of record pairs can be selected on the 'Comparison' tab, as shown in Figure 6. *Febrl* contains 26 similarity functions, including 20 approximate string comparison functions [3], as well as functions specialised for dates, times, ages, or numerical values. All these similarity functions return a numerical value between 0 (total dissimilarity) and 1 (exact match). It is possible to adjust these values by setting agreement and disagreement weights, as well as a special value that will be returned if one or both of the compared values is/are empty. The similarity weights calculated for each compared record pair will be stored in a weight vector, to be used for classifying record pairs in the next step.

2.6 Weight Vector Classification

Febrl contains several record pair classifiers, both supervised and unsupervised techniques. The traditional 'FellegiSunter' classifier requires manual setting of two thresholds [13], while with the supervised 'OptimalThreshold' classifier it is assumed that the true match status for all compared record pairs is known, and thus an optimal threshold can be calculated based on the corresponding summed weight vectors. Another supervised classifier is 'SuppVecMachine', which implements a support vector machine.


Febrl - (None)*
File Tools Help
Execute New Open Save Quit
Data Explore Index Compare Classify Output/Run Log

Standardisation [✓ Linkage] Deduplication Geocoding

Weight vector classification method: TwoStep

Match comparison value: 1.0 Match method: Nearest Selection nearest: 100 [✓] Select unique weight vectors

Non-match comparison value: 0.0 Non-match method: Nearest Selection nearest: 300 [✓] Select unique weight vectors

Random selection method: (None)

Step two classification method: SVM Kernel type: RBF C: 10

Generated Febrl Python code for classification (see Log page for generated code).

Figure 7: Example 'Two-Step' unsupervised weight vector classifier.

Both the 'KMeans' and 'FarthestFirst' [14] classifiers are unsupervised clustering approaches, and group the weight vectors into a match and a non-match cluster. Various centroid initialisation and distance measures are implemented in *Febrl*. Finally, the 'TwoStep' classifier, shown in Figure 7, is an unsupervised approach which in a first step selects weight vectors from the compared record pairs that with high likelihood correspond to true matches and true non-matches, and in a second step uses these vectors as training examples for a binary classifier [5, 7, 6].

2.7 Output Files and Running a Project

On this tab (not shown due to space limitation) the user can select various settings of how the match status and the matched record pairs will be saved into files. With a click on 'Execute', the *Febrl* GUI will ask the user if the generated project should be saved as a Python file, and if the project should be started and run within the GUI. Once started, a window will pop up showing a progress bar.

2.8 Evaluation and Clerical Review

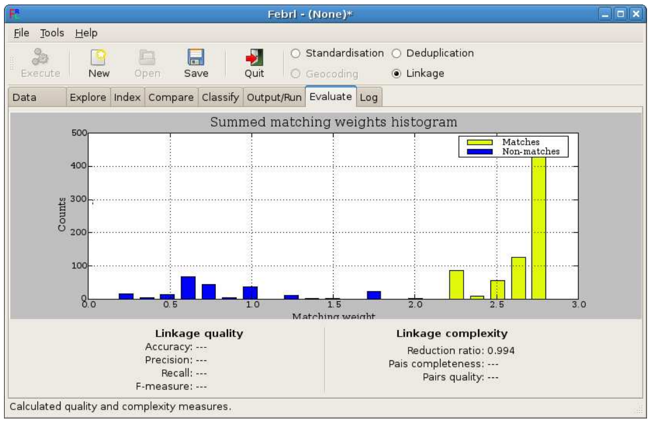
On the 'Evaluation' tab, shown in Figure 8, the results of a deduplication or linkage project are visualised as a histogram of the summed matching weights of all compared record pairs. If the true match and non-match status of record pairs is available, the quality of the conducted linkage will be shown using the measurements of accuracy, precision, recall and F-measure. Also shown are measures that allow the evaluation of the complexity of a deduplication or linkage project (i.e. the number of record pairs generated by the indexing step and their quality), these measures are the reduction ratio, pairs completeness and pairs quality [10].

2.9 Log Tab

On this tab the *Febrl* Python code generated throughout a project is shown, allowing experienced users to verify the correctness of the generated code. It also enables copying of this code into a user's own *Febrl* Python modules.

3. ACKNOWLEDGEMENTS

This work is supported by an Australian Research Council (ARC) Linkage Grant LP0453463 and partially funded by the New South Wales Department of Health, Sydney.


Febrl - (None)*
File Tools Help
Execute New Open Save Quit
Data Explore Index Compare Classify Output/Run Evaluate Log
Summed matching weights histogram
Matches
Non-matches
Counts
Matching weight
0
100
200
300
400
500
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Standardisation [✓ Linkage] Deduplication Geocoding

Linkage quality

Accuracy: ---
Precision: ---
Recall: ---
F-measure: ---

Linkage complexity

Reduction ratio: 0.994
Pais completeness: ---
Pairs quality: ---

Calculated quality and complexity measures.

Figure 8: Evaluation tab showing the matching weight histogram and quality and complexity measures for a linkage.

4. REFERENCES

[1] A. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *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 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 *MCD'06, held at IEEE ICDM'06*, Hong Kong, 2006.

[4] P. Christen. Towards parameter-free blocking for scalable record linkage. Technical Report TR-CS-07-03, The Australian National University, Canberra, 2007.

[5] P. Christen. A two-step classification approach to unsupervised record linkage. In *AusDM'07*, pages 111–119, Gold Coast, Australia, 2007.

[6] P. Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *ACM SIGKDD'08*, Las Vegas, 2008.

[7] P. Christen. Automatic training example selection for scalable unsupervised record linkage. In *PAKDD'08, Springer LNAI 5012*, pages 511–518, Osaka, Japan, 2008.

[8] P. Christen. Febrl - A freely available record linkage system with a graphical user interface. In *HDKM'08, CRPIT vol. 80*, pages 17–25, Wollongong, Australia, 2008.

[9] P. Christen and D. Belacic. Automated probabilistic address standardisation and verification. In *AusDM'05*, Sydney, 2005.

[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*. Springer, 2007.

[11] 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.

[12] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM SIGKDD'02*, Edmonton, 2002.

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

[14] K. Goiser and P. Christen. Towards automated record linkage. In *AusDM'06*, pages 23–31, Sydney, 2006.

[15] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD'95*, pages 127–138, San Jose, 1995.

[16] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA'03*, Tokyo, 2003.

[17] G. J. Williams. Data mining with Rattle and R. Togaware, Canberra, 2008. Software available at: `http://datamining.togaware.com/survivor/`.