Accurate Synthetic Generation of Realistic Personal Information

Peter Christen[1] and Agus Pudjijono[2]

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

[2] Data Center, Ministry of Public Works of Republic of Indonesia
Jakarta, 12110, Indonesia,
apudjijono@pu.go.id

Abstract. A large portion of data collected by many organisations today is about people, and often contains personal identifying information, such as names and addresses. Privacy and confidentiality are of great concern when such data is being shared between organisations or made publicly available. Research in (privacy-preserving) data mining and data linkage is suffering from a lack of publicly available real-world data sets that contain personal information, and therefore experimental evaluations can be difficult to conduct. In order to overcome this problem, we have developed a data generator that allows flexible creation of synthetic data containing personal information with realistic characteristics, such as frequency distributions, attribute dependencies, and error probabilities. Our generator significantly improves earlier approaches, and allows the generation of data for individuals, families and households.

Keywords: Artificial data, data matching, data linkage, privacy, data mining pre-processing.

1 Introduction

Today, massive amounts of data are collected by many organisations in both the private and public sectors. A large proportion of this data is about people, and often personal identifying details are stored together with application specific information, for example employment or medical details. When such data is analysed within an organisation, then normally, depending upon the desired outcomes, only parts of the personal information is used for an analysis (like age, gender, or postcode). In these cases, privacy and confidentiality are generally not of great concern, as the results of the analysis are only used within an organisation, and no detailed private or confidential information is released.

However, when data is being shared between organisations, privacy and confidentiality become of paramount importance, because personal information is commonly required to match records from different databases [1]. The aim of such linkages is to match all records that refer to the same entity. Because real-world data is commonly dirty [2] (contains errors and variations) and often no unique entity identifiers are available, sophisticated approximate matching algorithms are required that use the available personal identifiers [3,4].

The process of *data linkage* or *matching* has in the past decade been recognised as an important and challenging problem, and a variety of novel linkage algorithms have been developed [3,4]. They mainly address the technical challenges of matching accuracy and scalability to very large databases. Another challenge for data linkage research is the lack of publicly available real-world test data sets that allow evaluation of new algorithms. This lack is due to privacy concerns, because it is illegal in most countries to publish data that contains, for example, personal details of customers or patients. As a result, data linkage researchers have to use publicly available data sets, or use their own (confidential) data, which prevents others from repeating experimental studies [5].

An alternative is to use synthetically generated data. This approach has several advantages. First, a user can control the size (number of records) and quality (error characteristics) of the generated data sets. Second, such data can be published, and thus allows other researchers to repeat experiments and better evaluate algorithms. Third, the generator itself can be published, allowing others to generate data that is specifically tailored to their use, for example to their country or application domain. Fourth, because it is known which of the generated records are matches, it is possible to calculate matching rates [3].

Besides data linkage research, any application area where data containing personal information is required for research purposes can benefit from synthetically generated data, because such data removes privacy and confidentiality concerns. Examples include research into privacy-preserving data sharing [1], publishing and mining, or statistical micro-data confidentiality.

The challenges when generating synthetic data are that it is not easy to create data with characteristics that are similar to real-world data. The frequency and error distributions of values have to follow real-world distributions, and dependencies between attributes have to be modelled. This paper describes a data generator with such characteristics. It is a significant improvement over earlier generators [2,5,6], which created data in less realistic ways.

2 Synthetic Data Generation

As illustrated in Fig. 1, the data generator works in two steps. First, a user specified number of *original* records is created based on real values and their frequencies and dependencies, or using specific attribute generation rules [5,7]. Second, randomly selected original records are modified into *duplicate* records. Alternatively, family and household records can be generated. As can be seen in Fig. 4, each record is given a unique identifier (*'rec_id'*) that will facilitate the calculation of matching rates [3].

2.1 Original Record Generation

For original records, the values in name and address attributes are created randomly using frequency tables. Such tables can, for example, be extracted from

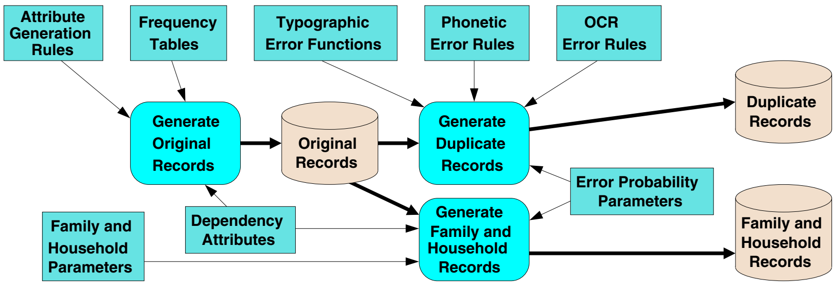
Attribute Generation Rules
Frequency Tables
Typographic Error Functions
Phonetic Error Rules
OCR Error Rules
Generate Original Records
Original Records
Generate Duplicate Records
Duplicate Records
Error Probability Parameters
Generate Family and Household Records
Family and Household Records
Family and Household Parameters
Dependency Attributes

Fig. 1. Overview of the data generation process

telephone directories. For date, telephone and social security number attributes, a user can specify generation rules that determine the range of dates (such as start and end birth dates), or the number of digits (for example for telephone numbers). In the following, we describe the two major novel features of our data generator [7]: family and household data, and attribute dependencies.

Generating Family and Household Data. The records for a family are generated by first selecting an original record at random. According to its age and gender values, it is assigned one of the roles *husband*, *wife*, *son* or *daughter*. The next step is to randomly select how many other records are to be generated for this family. These records are then created by keeping the surname of the first family record, but modifying given name, gender and age values. Address attribute values are generally kept the same for all members of a family. Depending upon the age of son and daughter records, however, a new address will be created with a certain probability, assuming the child has left home.

Household records are generated similarly, with the main difference being that all records in a household have different names but the same address, and that all age values are above 18 (one of many parameters that a user can set [7]).

Attribute Dependencies. A dependency occurs if the values in an attribute depend upon the values in one or more other attributes. For example, given names depend on the gender and the cultural background of a person, while suburb/town names depend on the state/territory they are located in. These dependencies are based on frequency tables, such as the one shown in Fig. 2.

When generating the original records, the *key* attributes (the attributes that others depend on) are generated first, and according to a selected key attribute value, a value from the dependent attribute is randomly chosen according to the corresponding frequency distribution. For example, using the values from Fig. 2, if the state *'QLD'* has been selected, the suburb name *'Allansford'* would be chosen with likelihood 6.25%, *'Allendale'* with likelihood 68.75%, and *'Allestree'* with likelihood 25%. To introduce randomness into the data, with a certain likelihood, as set by the user, a dependency is not followed, but rather a value is randomly chosen from the overall frequency table of the dependent attribute.

```
ACT  :  Acton;5, Ainslie;10, Amaroo;7, Belconnen;12
NSW  :  Albanvale;3, Albert Park;6, Alberton;4, Albion Park;9
QLD  :  Allansford;1, Allendale;11, Allestree;4
```

Fig. 2. Sample from a combined dependency–frequency look-up table with Australian state names on the left, and suburb names and their frequencies on the right

2.2 Error Modelling and Duplicate Record Modification

As illustrated in Fig. 3, data can be entered through a variety of channels, each having its own error characteristics. For example, handwritten forms that are processed using optical character recognition (OCR) software will likely include substitutions between similar looking characters. On the other hand, phonetic errors, like the variations *'Dickson'* and *'Dixon'*, are introduced when information is dictated using speech recognition, or typed manually. Typing itself introduces certain errors more likely than others. Depending upon keyboard layout, mistyping neighbouring keyboard keys, such as *'a'* and *'s'*, can occur. Often, depending upon the data entry channel, a combination of error types is introduced.

Our data generator can model typographic, phonetic and OCR errors. For each error type, a user can set how likely they are introduced when the duplicate records are generated. Setting the likelihood of typographic and phonetic errors to 0, for example, will result in duplicate records that only contain OCR errors.

Typographic Errors. These errors include insertion, deletion, and substitution of a character; and transposition of two adjacent characters. They are implemented as functions that apply the corresponding modification to a given input string with a certain likelihood (as set by the user), and return the modified string. Following studies of error distributions [8], the position of a modification is randomly chosen such that it more likely occurs in the middle or towards the end of a string, because real errors are less likely at the beginning of names.

Optical Character Recognition Errors. OCR modifications are based on rules that consider shape similarity among characters, such as *'5'* and *'S'* or *'w'* and *'vv'*. Around fifty such rules are used, representing the most likely OCR variations that might occur. When duplicate records are generated, one or more possible OCR modifications will be randomly selected and applied to an input string, and the modified string will be inserted into the duplicate record. Like phonetic errors, OCR errors can be a single character modification or a combination of modifications (like a substitute and delete, or a delete and insert).

Phonetic Errors. These errors are usually more complex than typographic or OCR errors, as they often include changes of character groups and depend upon the position within a string. The idea behind our approach in modelling phonetic errors is to employ the rules that are used in phonetic encoding methods [9], such as *Phonix* and *Double-Metaphone*. In encoding methods, such rules are used to

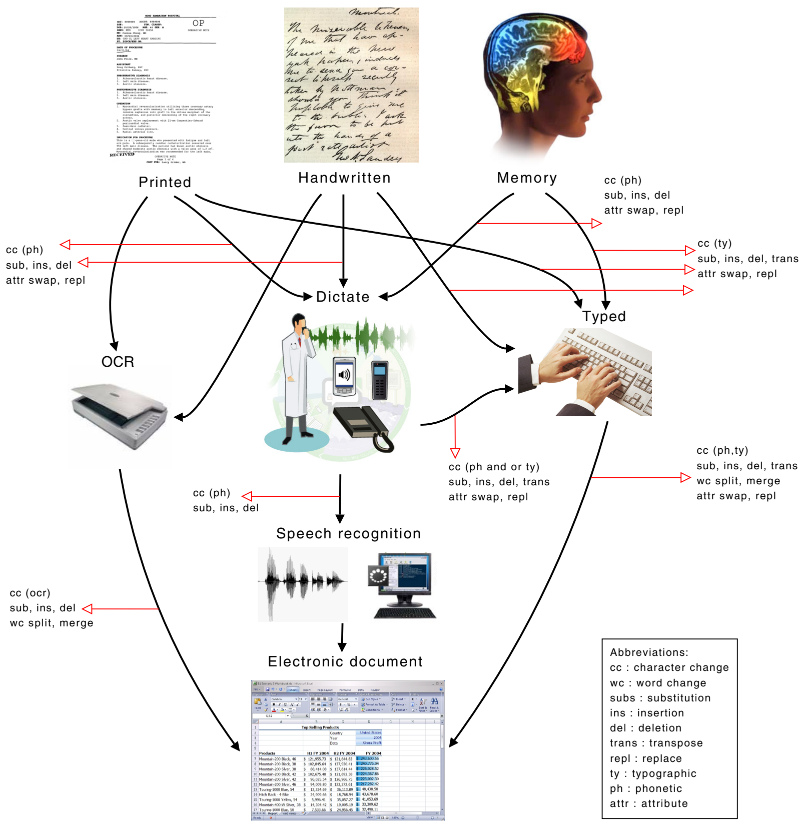

Fig. 3. Model of data sources and possible errors introduced during data entry

group similar sounding names together, while we employ them to modify a name in order to generate a similar sounding variation of it. Currently, around 350 phonetic rules are used, each made of the following seven components.

1. **Position.** The position within the input string where the *original* string pattern can occur. The four possible values are: *ALL* (can occur anywhere), *START* (must occur at the beginning), *MIDDLE* (must occur in the middle, but not at the beginning or end), and *END* (must occur at the end).
2. **Original pattern.** This is the string (made of one or more characters) that will be replaced with the *substitute* string pattern if the rule is applied.
3. **Substitute pattern.** This is the string (made of zero or more characters) that will replace the *original* string pattern if the rule is applied.
4. **Precondition.** A condition that must occur before the *original* string pattern in order for this rule to become applicable. A precondition can be that the character immediately before the original pattern is a vowel (‘*V*’), a consonant (‘*C*’), or a more complex expression [7].

5. **Postcondition.** Similarly, a condition that must occur after the *original* string pattern in order for this rule to become applicable.
6. **Pattern existence condition.** This condition requires that a certain given string pattern does (*'y'* flag) or does not (*'n'* flag) occur in the input string.
7. **Start existence condition.** Similarly, this condition requires that the input string starts with a certain string pattern (*'y'* flag) or not (*'n'* flag).

The last four components of a rule (its conditions) can be set to *'None'* if they are not required. In the following we give two illustrative examples.

– `ALL, h, @, None, None, None, None` *(mustap<u>h</u>a → mustapa)*
  In this rule there are no conditions, so any occurrence of the character *'h'* is being removed (replaced with an empty string – denoted with a *'@'*).
– `END, le, ile, C, None, None, None` *(bramb<u>le</u> → bramb<u>ile</u>)*
  The precondition in this rule is *'C'*, which means the character before the original pattern must be a consonant. The character before the pattern *'le'* in this example is a *'b'*, so the modification *'le'* into *'ile'* is applied.

3 Experimental Evaluation

Our data generator is implemented as part of the *Febrl* (Freely Extensible Biomedical Record Linkage) open source data linkage system [10],[1] and is written in the *Python* programming language. Due to the availability of its source code, it can be modified and extended according to a user's needs. A large number of parameters can be set by the user, including the number of original and duplicate records to be generated, the frequency and dependency look-up tables to be used, the distributions used for household and family records, and the various error characteristics to be applied when duplicates are created [7].

We used a variety of data sets to create our look-up tables, including a data set containing 99,571 names and their culture of origin (37 different cultures) [11], a data set with Australian postcode, suburb and state values as available from the *Australia Post* Web site,[2] and the various look-up tables supplied with the *Febrl* data linkage system [10]. Error and modification probabilities were set according to real world studies on typographic and other errors [8,12,13,14].

Some example data that was created using our generator is shown in Fig. 4. As can be seen, the record identifiers (*'rec_id')* designate if a record is an original or a duplicate, and the duplicate records are numbered and refer back to their original record, in order to allow the calculation of matching rates [3].

We have conducted a large number of experiments to validate our data generator, by comparing real data sets with synthetic data that was generated using frequency tables based on the real data. A detailed analysis and discussion is provided elsewhere [7]. A user can repeat our experiments by downloading the *Febrl* system and run the generator program supplied with it (and possibly change parameter settings according to her or his needs).

[1] Available from: `https://sourceforge.net/projects/febrl/`
[2] Available from: `http://www.post.com.au/postcodes/`

rec_id	age	given_name	surname	street_number	address_1	address_2	state	suburb	postcode
rec-1-org	***33***	***Madison***	Solomon	***35***	Tazewell ***Circuit***	Trail View	***VIC***	***Beechboro***	***2761***
rec-1-dup-0	33	Madisoi	Solomon	35	Tazewell Circ	Trail View	VIV	Beech Boro	2761
rec-1-dup-1		Madison	Solomon	36	Tazewell Crct	Trail View	VIC	Bechboro	2716
rec-2-org	39	***Desirae***	***Contreras***	44	Maltby Street	***Phillip Lodge***	NSW	***Burrawang***	3172
rec-2-dup-0	39	Desirae	Kontreras	44	Maltby Street	Phillip Loge	NSW	Burrawank	3172
rec-2-dup-1	39	Desire	Contreras	44	Maltby Street	Fillip Lodge	NSW	Buahrawang	3172
rec-3-org	***81***	***Madisyn***	Sergeant	6	***Howitt*** Street	Creekside Cottage	VIC	***Nangiloc***	3494
rec-3-dup-0	87	Madisvn	Sergeant	6	Hovvitt Street	Creekside Cottage	VIC	Nanqiloc	3494

Fig. 4. Examples of generated data. Records '1' include typographical errors, records '2' phonetic errors, and records '3' OCR errors. The original values that were modified are in bold-italics, and their corresponding modifications are underlined.

4 Related Work

A first data generator for personal information was developed in the 1990s [2]. It allowed the generation of data based on lists of names, cities, states and postcodes, however without using any frequency distributions. A user could set the size of the data sets to be generated, and the types and amount of errors to be introduced. An improved generator was described more recently [6]. It allowed attribute values to become missing, and it improved the variability of the created values. It is however unclear if this generator is using frequency information, as not many details have been described.

A first simple version [5] of our generator has been freely available as part of the *Febrl* [10] data linkage system. It improved upon earlier generators by including frequency tables of attribute values, more flexible setting of individual error probabilities, as well as inclusion of look-up tables with name variations (to be used for example for nick-names, known phonetic variations, and common misspellings). This generator however does not include attribute dependencies, does not allow creating family or household record groups, and it does not model errors as accurately as the new version described in this paper.

The phonetic error model presented in Sect. 2.2 is based on rules that were originally developed for phonetic encoding methods [9]. A common feature of phonetic encodings is that they convert name strings into codes according to how a name is being pronounced. Names that sound similar are converted into the same code. This is obviously a language dependent process, and most phonetic encoding methods have been developed for the English language.

Our work is also based on various studies that have analysed spelling and data entry errors and their corrections [8,12,13,14]. These studies found that most errors are single character errors, and that the distribution of error types depends upon the mode of data entry. For example, OCR output contains almost exclusively substitution errors, while this type of error accounts for less than 20% of errors with keyboard based manual data entry [8]. Typically, up to 95% of misspellings in keyboard entry only contain one error; with only around 8% of first letters incorrect, compared to almost 12% of second and nearly 20% of third letters.

5 Conclusions

We have presented a data generator for personal information that allows the generation of realistic synthetic data based on frequency tables and attribute generation rules. There are various ways to improve our generator. First, allowing the generation of not just personal information, but also application specific attributes (like medical, employee, or customer details) will make our generator applicable to the wider data mining community. Second, extending family records to include other roles (such as cousins, aunts, uncles, etc.) and allow culture specific parameter settings will enable the generation of the complex family connections that occur in real life. Third, enabling Unicode characters will make our generator more international and will allow the generation of data sets containing, for example, Thai, Chinese, or Arabic characters. Finally, adding a graphical user interface will facilitate the setting of the many possible parameters. Another part for our future work will be to fully integrate our new data generator into our *Febrl* data linkage system [10].

References

1. Christen, P.: Privacy-preserving data linkage and geocoding: Current approaches and research directions. In: ICDM PADM workshop, Hong Kong (2006)
2. Hernandez, M., Stolfo, S.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery 2(1), 9–37 (1998)
3. Christen, P., Goiser, K.: Quality and complexity measures for data linkage and deduplication. In: Quality Measures in Data Mining. Studies in Computational Intelligence, vol. 43, pp. 127–151. Springer, Heidelberg (2007)
4. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1), 1–16 (2007)
5. Christen, P.: Probabilistic data generation for deduplication and data linkage. In: Gallagher, M., Hogan, J.P., Maire, F. (eds.) IDEAL 2005. LNCS, vol. 3578, pp. 109–116. Springer, Heidelberg (2005)
6. Bertolazzi, P., De Santis, L., Scannapieco, M.: Automated record matching in cooperative information systems. In: DQCIS, Siena, Italy (2003)
7. Pudjijono, A.: Probabilistic data generation. Master of Computing (Honours) thesis, Department of Computer Science, The Australian National University (2008)
8. Pollock, J., Zamora, A.: Automatic spelling correction in scientific and scholarly text. Communications of the ACM 27(4), 358–368 (1984)
9. Christen, P.: A comparison of personal name matching: Techniques and practical issues. In: ICDM MCD workshop, Hong Kong (2006)
10. Christen, P.: Febrl – An open source data cleaning, deduplication and record linkage system with a graphical user interface. In: ACM KDD, Las Vegas (2008)
11. Phua, C., Lee, V., Smith-Miles, K.: The personal name problem and a recommended data mining solution. In: Encyclopedia of Data Warehousing and Mining, 2nd edn., Information Science Reference (2008)
12. Damerau, F.: A technique for computer detection and correction of spelling errors. Communications of the ACM 7(3), 171–176 (1964)
13. Hall, P., Dowling, G.: Approximate string matching. ACM Computing Surveys 12(4), 381–402 (1980)
14. Kukich, K.: Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), 377–439 (1992)