Project N°: 262608

ACRONYM: Data without Boundaries

Data without Boundaries

Deliverable D11.2

Record Linkage Approaches for Dynamic Database Integration

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 262608 (DwB - Data without Boundaries).

Privacy preserving record linkage

D. Smith and N. Shlomo

School of Social Sciences
Humanities Bridgeford Street
University of Manchester
Oxford Road
Manchester
M13 9PL, UK

2014-03-07

1 Introduction

Data on individuals is often contained in distinct databases held by distinct organisations. In the absence of unique identifiers (such as NHS number) record linkage can be used to join the data and create combined datasets for research and other purposes. This is common with medical databases and will become more common for administrative databases given current plans for abolition of the UK Census beyond 2011.

Data may be held under a variety of data sharing agreements. There might be privacy guarantees that prevent the release of certain data for linkage, or any other purposes. A common practice is to suppress information that might directly identify an individual. Fortunately non identifying data can be useful for research purposes. Unfortunately it is the identifying information that is most useful for record linkage. For this reason techniques have been developed to anonymize data in such a way that they can still be used for linkage. Variables common to distinct databases that are used for data linkage are termed *key variables*.

The basic scenario is that databases are held in geographically distinct locations by Alice and Bob. They contain common variables that contain identifying information (name, address etc.). The information on key variables is anonymized before passing to a third party, Carol, who performs the record linkage. Typically Carol would be supplied with only this data and (randomly assigned) unique keys for the records supplied by Alice and Bob. Carol performs the record linkage and returns the results as pairings on the unique keys. This allows the merging of the data held by Alice and Bob so that it can be released after suppression of the identifying variables.

There are three important aspects to the process:

Firstly, the data held by Alice and Bob need to be cleaned and harmonized. Cleaning would typically involve such things as correcting obvious typographical errors and removing leading / trailing white space from records. Harmonization is intended to make the databases comparable on their fields. This might involve string substitutions so that, for example, both databases use 'm' and 'f' to denote gender. Harmonization might also involve breaking up fields into component fields. This is common with dates, and can promote better record linkage.

Secondly, the data released to Carol need to be adequately protected. It should not be easy for Carol to identify the underlying strings. One issue, that is relatively easily avoided is a dictionary attack. If the anonymization scheme is deterministic and can be emulated by Carol, then she can create a mapping of anonymized values to likely strings. A less easily avoided issue is that of a frequency attack. Frequently occurring strings will generate frequently occurring anonymized values. The number of distinct values for a field will provide a clue as to the variable, two distinct values perhaps being suggestive of gender. If there is some reason to expect that the databases contain more males than females, then Carol can determine gender with some degree of confidence. The risk from frequency attacks cannot be completely eliminated, as identifying matching variable values is the basis of record linkage. For instance, attempting to disguise the fact that a variable can only take two values would have a large impact on the record linkage process.

Thirdly, we have the record matching on anonymized data. Exact and probabilistic approaches are possible. The exact approach requires an exact match on all key variables in order to classify a pair of records as a true link. Probabilistic approaches allow for some degree of error on key variables. They might exploit similarity measures on pairs of strings. This has been shown to improve the record linkage process appreciably (e.g. Schnell et al., 2009). One aspect of record linkage that is all but impossible using anonymized data is *clerical review*. Possible linkages that are questionable are examined by a human who can make the final decision as to whether the link is correct or not. Anonymization disguises the very information that would be useful in this process. So in a privacy preserving context it is generally assumed that clerical review is not possible - possible linkages must be dichotemised into true and false links.

This report considers the second and third aspects of privacy preserving record linkage - string anonymization, and record linkage using similarity scores. These are relatively distinct problem areas. The former concerns the transformation of strings into a secure format that can be used (by Carol) to generate (or estimate) similarity scores on the underlying strings. The latter concerns the effective use of similarity scores in record linkage. In a non privacy preserving context (trusted Carol) there is a greater degree of flexibility in similarity score generation. A wider range of scores are available, and they can be calculated exactly from the underlying strings. In this context it is also possible for Carol to perform clerical review. However, this is still an issue for *on the fly* record linkage, so this report places emphasis on the automatic handling of questionable links. It also emphasises the importance of satisfactorily handling missing values.

Section 2 provides some background on existing string anonymization approaches, and describes an alternative that is very robust to frequency attacks on tokens. Implementation details are provided. Section 3 details an existing Expectation-Maximization approach to record linkage, before presenting an extension to the method which offers a number of advantages. The important topics of blocking and threshold selection are covered in Subsection 3.5. The application of existing approaches is discussed, and details of alternatives / extensions to these approaches are presented in appendices. Section 4 contains the results of record linkage experiments. The work is summarised in Section 5.

2 Privacy preserving string anonymization

The simplest approach is to use a cryptographic hash function such as MD5[1] to anonymize values. Hash functions convert strings to integers. Equal strings will produce equal hash values. MD5 hashing will produce the same results on all computer platforms. Subsequent linkage can be based on the equality or inequality of integers (hash values) rather than strings. For cryptographic hashes it is generally infeasible to recover the underlying string from the hash. However, it is easy to hash candidate strings until a matching hash value is produced (a dictionary attack). Alternatively, non-deterministic alternatives such as HMAC[2] can be used. These require Alice and Bob to use a common secret key. But the underlying problem that remains is that values can only be compared on the basis of equality or inequality. Generally, data cleaning and harmonization cannot guarantee that the values for a given individual on a common variable will be identical. Errors will remain, and when these are typographical errors or alternative spellings for words, then the degree of similarity between strings can be useful for record linkage. This is particularly the case when there is no opportunity for clerical review. Thus investigators have produced methods for anonymizing strings that allow similarity scores to be produced.

Churches and Christensen (2004) used a scheme where Alice and Bob would split the strings into bigrams before hashing. Carol would be supplied with a set of hashed bigrams for each key variable for each record.

$$\text{'john'} \rightarrow \{\text{'jo', 'oh', 'hn'}\} \rightarrow \{21299418, 21496024, 20971735\}$$
$$\text{'jon'} \rightarrow \{\text{'jo', 'on'}\} \rightarrow \{21299418, 21889246\}$$

Hashing disguises the true values of the bigrams, but Carol can see how many bigrams match. This can be used to generate similarity measures such as the dice coefficient,

$$D_{A,B} = \frac{2|A \cap B|}{|A| + |B|}$$

[1]http://en.wikipedia.org/wiki/MD5
[2]http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

where A and B are the bigram hash sets for potentially matching strings from Alice and Bob respectively.

Simply supplying sets as illustrated above is vulnerable to a frequency attack by Carol. Common bigrams will result in common hashes. If Carol knows that bigrams have been used, then she can recover the lengths of the underlying strings. So further measures are generally required in order to adequately protect the data. Some of these, such as padding strings to make them all equal in length, are detailed in Churches and Christensen (2004).

There are alternative ways of *tokenizing* strings – such as unigrams (for short strings) or extended bigrams (trigrams with the central letter removed). Churches and Christensen present a protocol which involves hashing all the non-empty members of the powerset of each bigram set.

2.1 Bloom filters

Schnell et al. (2009) also use a scheme based on bigrams and the dice coefficient, but Alice and Bob supply Bloom filters[3] rather than hash sets. A Bloom filter is a probabilistic set data structure. A Bloom filter is a bitstring – an array of 0s and 1s, which can be represented as an integer (in its binary form). The number of bits, m, is fixed, and k hash functions are used to map an item to k array indices in the range [0, m-1]. Adding an item to the Bloom filter is simply the process of setting the bits at the relevant indices to 1. Set membership can be tested by generating the indices for an item and checking if all the relevant bits are set. If any are 0, then the item is not in the set. If all are 1, then the item might be in the set (the bits might have been set on the addition of other items). The sizes of the sets A and B and their intersection can be approximated from two Bloom filters, allowing the dice coefficient to be approximated,

$$D_{A,B} = \frac{2h}{(a+b)}$$

where h is the number of bits set to 1 in both bitstrings, and a and b are the numbers of bits set to 1 in A and B respectively.

Schnell et al. (2009) discuss the possibility of launching a frequency attack against a set of Bloom filters and measures that can be taken to mitigate the risk. These measures include adding dummy strings / Bloom filters, and adding random q-grams to Bloom filters or randomly setting bits (to 1). Swamidass and Baldi (2007) provide an estimator for the number of items in a Bloom filter.

2.2 Locality sensitive hashing

Locality sensitive hash functions generate hashes so that similar strings produce similar hashes. This is not dissimilar to the Bloom filter approach, as an m-bit Bloom filter can be represented as an m-bit integer.

Minwise hashing (Broder, 1997) is a method that generates a random permutation of a set of elements and returns the hash for the first ordered element. In order to generate a consistent permutation across a number of sets the number of possible elements must be finite. This universe of elements can be permuted, and distinct sets of elements reordered according to the permutation of the universe. If we consider the union of two sets A and B, then the probability that any given element in the union is ordered first is simply $1/|A \cup B|$. If this element is common to both sets, then the hashes will collide (be equal). Thus the probability of a hash collision is equal to the Jaccard similarity measure,

$$J_{A,B} = \frac{|A \cap B|}{|A \cup B|}.$$

In order to be able to estimate the Jaccard score from minwise hashes, many hashes must be produced.

A practical difficulty with the permutation generation can be the size of the universe. For a given tokenization scheme it might be manageable, but ideally we would not want to be limited to a given scheme. Alternative schemes might suit different types of field, for instance numerical data. For this reason hashing is often used to generate the permutation.

If we use a suitable hash function to hash the elements and then reorder by hash value, then we have generated a random permutation of the elements. The hash of the first ordered item in each set is the (minwise) hash value for the set. Of course, this is the same as generating the hash values for the tokens in a set and returning the minimum hash value. In practice, this is how minwise hashing is often implemented.

[3] http://en.wikipedia.org/wiki/Bloom_filter

In order to generate an estimate of the Jaccard score we need to generate many hash values using many hash functions. Then, the number of collisions is distributed,

$$n_z \sim Bin(m, J_{A,B})$$

where m is the number of hash functions used.

As the number of collisions follows a Binomial distribution we can define m to be an integer large enough to produce an estimate of $J_{A,B}$ to a given level of precision. The estimator and its variance are simply,

$$\widehat{J}_{A,B} = \pi$$

where $\pi = n_z/m$ is the proportion of collisions, and,

$$Var(\widehat{J}_{A,B}) = \frac{J_{A,B}(1 - J_{A,B})}{m}.$$

2.3 Minwise hash implementation

It has been demonstrated that a number of minwise hashes can be used to generate an estimated Jaccard score. For each string a list of hash values is generated. To compare strings their lists of hash values are compared and the proportion of collisions is the Jaccard score estimate. So if Alice and Bob use the same hash functions, then they can independently generate hash lists that can be used by Carol for string comparison. We need to identify a suitable family of hash functions for minwise hashing.

Wegman and Carter (1981) define a hash function as being k-independent if for any distinct x_1, ..., x_k keys the values $h(x_1)$, ..., $h(x_k)$ are independent random variables uniformly distributed over the relevant range. Simple tabulation hashing has been shown to be only 3-independent. But this is not the form of independence we require for minwise hashing.

Consider a set S of keys and $x \notin S$. Then minwise independence holds if,

$$Pr[h(x) < min h(S)] = \frac{1}{|S| + 1}.$$

Patrascu and Thorup (2011) show that simple tabulation hashing is suitable for minwise hashing.

2.3.1 Simple tabulation hashing

Simple tabulation hashing works by splitting a fixed length binary key into c characters. For each character position a lookup table of random integers is generated, so that each distinct character in that position can be used to look up a random integer. These are looked up for each character in a string and XORed to produce a hash for the key.

For example the binary key,

10001010101010110111010101010011

can be split into four 8-bit keys.

10001010, 10101011, 01110101, 01010011

Each key is used to look up a random integer in a separate lookup table containing 2^8 randomly generated integers. Thus we need 4×2^8 random q-bit integers to generate a q-bit hash. The minwise independence assumption is more reasonable for small c (see Patrascu and Thorup, 2011), although this requires the generation of larger numbers of random integers.

As we are hashing variable length strings we hash them to an intermediate form that is suitable for tabulation hashing. The current implementation uses the 32 least significant bits of an MD5 hash. Four lookup tables are used, each containing 256 64-bit random integers.

Generating m hash functions simply requires the generation of m sets of lookup tables. There are several parameters that can be changed to trade off computational cost against the precision of Jaccard score estimates. The alternatives are still to be investigated. A variation of simple tabulation hashing, named twisted tabulation hashing, is even better for minwise hashing (Mikkel Thorup – personal communication).

	H_1	H_2	H_3	H_4	H_5	···	H_m
S_1	451153726	1123790273	2501120381	2030682762	965995804		
S_2	797504823	1123790273	262296169	1744666338	965995804		
···							
S_n							

Table 1: Minwise hashes for token sets S1={'jo', 'oh', 'hn'} and S2= {'jo', 'on'}

Minwise hashing produces a list of hashes for each record (and key variable). Table 1 shows simple tabulation minwise hashes for the bigram-tokenized strings 'john' and 'jon'. The common bigram 'jo' results in two hash collisions - each hash corresponding to the hash of a single token.

The true Jaccard score is 1/4. The Jaccard score estimate based on the first 5 hash functions is 2/5. Of course the variance for this estimate is large given only 5 hash functions.

It is clear that minwise hashes are vulnerable to a frequency attack. In fact, Carol has m distinct hash functions to attack. An additional issue is that the minwise hash values are essentially order statistics. Large token sets will tend to generate lower minwise hashes. If Carol was aware of the tokenization scheme (or simply assumed that longer strings would produce larger token sets), then she could make inferences about the length of the underlying strings – and perhaps identify the variable.

The next sections outline a proposed approach to practically eliminate the security issues surrounding Bloom filters and minwise hashes.

2.3.2 b-bit hashing

An alternative to returning the full hash value for minwise hashing is to return a fixed number of bits, say the b least significant bits. Li and König (2011) use b-bit hashing to reduce storage costs, their application being the hashing of large text corpora. For our purposes b-bit hashing can be used to mitigate against frequency attacks on the individual hash functions, and to disguise the sizes of the token sets. By restricting the number of published bits we purposely generate hash collisions that do not correspond to matching tokens.

The additional collisions require a new estimator for the Jaccard score. Earlier it was assumed that the probability of collisions other than for matching tokens was negligible. This is a reasonable assumption for token sets that are very much smaller in size than the number of possible hashes. The following adopts the same assumption.

If we only consider the b most significant bits, then the probability of a collision is 1 if the q-bit hashes collide. There are exactly 2^q possible full hashes with 2^{q-b} possible full hashes for each possible b-bit hash. For any b-bit hash there are thus 2^{q-b}-1 distinct q-bit hashes that do not match on all q bits. Thus the collision probability on the most significant b bits given distinct q-bit hashes is $(2^{q-b}$-1$)/2^q$. So the collision probability is,

$$Pr(z|q,b) = Pr(z|q) + (1 - Pr(z|q)) \times \frac{2^{q-b}-1}{2^q}$$

$$Pr(z|q,b) = Pr(z|q) + (1 - Pr(z|q)) \times (\frac{1}{2^b} - \frac{1}{2^q}).$$

For q sufficiently greater than b,

$$Pr(z) \approx J_{A,B} + (1 - J_{A,B}) \times 1/2^b$$

producing the resulting estimator,

$$\widehat{J}_{A,B} = \frac{\pi - (1/2)^b}{1 - (1/2)^b}$$

where π is the fraction (over m hash functions) of collisions on the least significant b bits.

This has variance,

$$Var(\widehat{J}_{A,B}) = \frac{\left(\frac{1}{2^b-1} + J_{A,B}\right)(1 - J_{A,B})}{m}.$$

2.3.3 Concatenated 1-bit minwise hashing

If we take Li and Konig's scheme to the extreme we can generate hashes on only the least significant bit. If we reproduce Table 1 but use 1-bit minhashes we get the table in Table 2.

	H_1	H_2	H_3	H_4	H_5	...	H_m
S_1	0	1	1	0	0		
S_2	1	1	1	0	0		
...							
S_n							

Table 2: 1-bit minwise hashes for token sets S1={'jo', 'oh', 'hn'} and S2= {'jo', 'on'}

The Jaccard score estimate (based on the first 5 hash functions) is now 3/5, and has higher variance than the estimate based on the full hashes.

We have generated two additional collisions. All values are 0 or 1, mitigating against frequency attacks on the columns, and there is no clue as to the length of the underlying strings (and no need to undertake measures such as string padding). Each row is essentially a random sequence of 0s and 1s.

For 1-bit hashing the estimator becomes,

$$\widehat{J}_{A,B} = 2\pi - 1$$

with variance,

$$Var(\widehat{J}_{A,B}) = \frac{1 - J_{A,B}^2}{m}.$$

Despite discarding all but 1 of the bits in the full hashes, the impact on the variance is relatively minor. 1 bit hashing inflates the variance by a factor,

$$\frac{Var_1(\widehat{J}_{A,B})}{Var_m(\widehat{J}_{A,B})} = 1 + \frac{1}{J_{A,B}}$$

which tends to 2 as $J_{A,B}$ tends to 1, but is only 3 for a Jaccard score as low as 0.5.

This reflects the main finding in Li and Konig (2011). They showed that Jaccard score estimates at a given level of precision could be generated using less storage if they generated more minwise hashes and discarded all but a subset of bits.

A list of m 1-bit hashes can be represented as a single m-bit hash. Each such hash is a random integer in the range [0, 2^m-1]. Any two hashes generated by token sets with empty intersection are statistically independent, and the expected proportion of collisions would be 0.5 – corresponding to a Jaccard score of 0.

The estimator can be re-expressed in terms of the hamming distance of the hashes,

$$\widehat{J}_{A,B} = 1 - \frac{2h_{A,B}}{m}$$

where $h_{A,B}$ can be calculated very efficiently using the method due to Wegner (1960).

As the estimated Jaccard score is a linear function of a Binomial proportion it is trivial to generate confidence intervals for $J_{A,B}$.

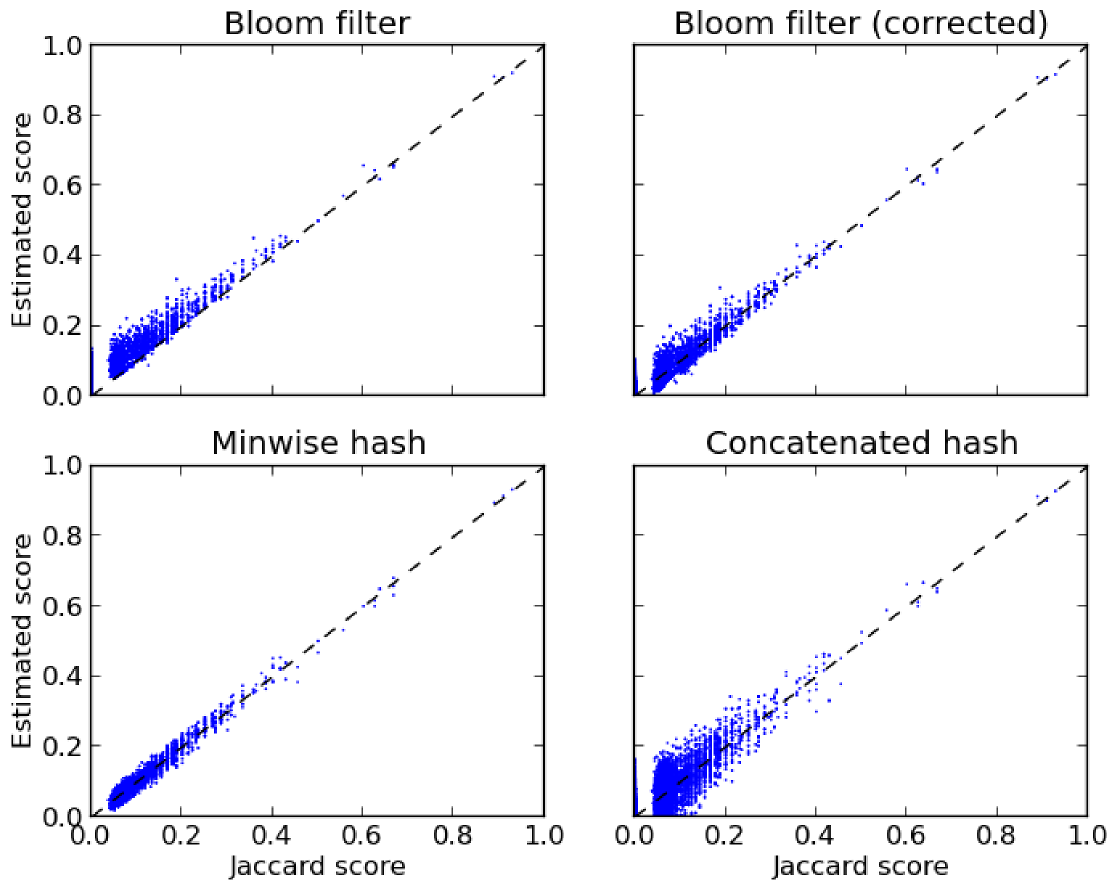

Figure 1: Estimated Jaccard scores against true Jaccard scores for alternative string anonymization schemes

2.3.4 Concatenated 1-bit hashing example

The string anonymization approaches described in this paper were compared using a random sample of (not necessarily distinct) 300 Titanic passenger surnames[4]. A second set of surnames was produced by perturbing these names to simulate typographical errors. All strings had leading and trailing underscores added before tokenizing into bigrams. Each exact string was compared with each perturbed string using the Bloom filter, minwise hashing and 1-bit hashing approaches. Estimated Jaccard scores were plotted against the exact Jaccard scores generated from the underlying token sets. The results are shown in Figure 1.

A 512-bit Bloom filter was used with 4 hash functions and the double hashing scheme due to Kirsch and Mitzenmacher (2006). Estimates from the Bloom filter were also generated using the corrected estimator due to Swamidass and Baldi (2007). The minwise hashing used the previously described scheme with 512 hash functions. Concatenated 1-bit hashes were generated by taking the least significant bits of the minwise hashes.

The biasedness of the (uncorrected) Bloom filter estimates is apparent. The minwise hash has smallest variance, but requires 64 times as much storage as any of the other approaches. The concatenated 1-bit hash estimates have largest variance, but are similar to the (uncorrected) Bloom filter estimates in terms of mean square error (due to the lack of bias).

It should be noted that the (uncorrected) Bloom filter estimates can exhibit large bias and large variance for poor choices of k. Choices designed to minimise the false positive rate (with respect to its use as a set data structure) tend to be very poor for Jaccard score estimation. (Here, several values of k were tried, and k=4 was found to be close to optimal.) The corrected Bloom filter estimates exhibit much lower bias. (The search for a good value for k showed that they are also much less influenced by choice of k). Nevertheless, choice of k is still an issue that needs to be addressed. This (and choice of size of Bloom filter) are affected by token set sizes. A choice that is optimal for small token sets might result in a large proportion of set bits for larger token sets.

[4]http://en.wikipedia.org/wiki/Passengers_of_the_RMS_Titanic

Minwise hashing and 1-bit concatenated hashing are unbiased (subject to the assumption of no unintended collisions on the full hashes), and we have closed form expressions for the variance of estimates. The variances depend only on the magnitude of the underlying Jaccard score and the length, m. Sizes of token sets will only impact the minwise hash based methods in so far as they influence the probability of collision on the full hashes (or intermediate representation given the described implementation). In all practical circumstances (at least for record linkage) the impact of collisions on the full hashes is vanishingly small. The only parameter to be decided is m, and this can be derived by specifying a required precision for a given Jaccard score threshold – scores above the threshold having precision strictly greater than the specified precision.

2.4 String anonymization summary

An import aspect of privacy preserving record linkage is the anonymization of variable values. This is generally performed by converting strings to integer values via hashing, or through the use of Bloom filters. Methods which use a single hash function to hash the underlying strings are not satisfactory as they only allow comparisons for equality. The performance of record linkage can be greatly improved if the similarity of strings is incorporated. Hashing individual tokens allows comparison of token pairs, and the generation of similarity measures. But this is less secure than simply hashing strings, and requires a carefully designed protocol to guard against attacks on the published data. Methods such as minwise hashing and Bloom filters allow the use of similarity scores for record linkage, but might still require additional measures to limit risks. It has been shown that these risks can be dramatically reduced via the simple approach of using minwise hashing, but discarding all but one bit of the minwise hashes (this does not actually need to be the least significant bit). This reduces the need for some of the perturbations that might used to reduce risks using other approaches. A 1-bit concatenated minwise hash is a random integer and provides no useful information regarding the size of the token set that produced it. The additional collisions that it generates provide a large degree of protection against frequency attacks on the tokens. Implementation is straightforward, efficient, and we have simple closed form expressions for the mean and variance of estimates.

No specific, detailed protocol has been presented. Existing approaches for sharing secret keys can be employed to share seeds for the generation of simple tabulation hash functions. The method can be plugged into existing protocols as an alternative to, for instance, Bloom filters. It simply provides a greater level of security, and the opportunity to remove steps that involve perturbation of the underlying strings or published bitstrings.

Concatenated 1-bit hashes generally require a little more storage (about 30% more) than Bloom filters to generate Jaccard scores to a given level of precision. Generation of 1-bit hashes tends to be more costly, due to the larger number of calls to hash functions for setting bits. Generation of similarity scores from concatenated 1-bit hashes tends to be a little more efficient than similarity score generation from Bloom filters. Despite the increased cost of generation, concatenated 1-bit hashes provide an attractive alternative to other methods.

3 Record linkage

We have two datasets A and B and seek to identify the records in A and B that correspond to the same population units. A and B are generally samples from a common population, so there is no 1 to 1 mapping from A to B.

We have the set of all possible matches.

$$A \times B = \{(a,b); a \in A, b \in B\}$$

This can be partitioned into sets of matched and unmatched pairs.

$$M = \{(a,b); a = b, a \in A, b \in B\}$$

$$U = \{(a,b); a \neq b, a \in A, b \in B\}$$

Each record is a vector of values on key variables, e.g. $a = \{a_1, \ldots, a_K\}$.

Fellegi and Sunter (1969) present a theory of record linkage. Their approach is essentially a Bayesian approach, although that is not the way in which they present it.

Bayes theorem leads to the following expression for the posterior odds that a and b correspond to the same population unit,

$$\frac{Pr((a,b) \in M|(a,b))}{Pr((a,b) \in U|(a,b))} = \frac{Pr((a,b)|(a,b) \in M)}{Pr((a,b)|(a,b) \in U)} \frac{Pr((a,b) \in M)}{Pr((a,b) \in U)}.$$

Fellegi-Sunter further assumes that the odds can be factorized as,

$$\frac{Pr((a,b) \in M|(a,b))}{Pr((a,b) \in U|(a,b))} = \left(\prod \frac{Pr((a_i,b_i)|(a,b) \in M)}{Pr((a_i,b_i)|(a,b) \in U)}\right) \frac{Pr((a,b) \in M)}{Pr((a,b) \in U)}.$$

This is a naïve Bayes assumption. Fellegi and Sunter define m and u probabilities as,

$$m = Pr(a_i = b_i|(a,b) \in M)$$

$$u = Pr(a_i = b_i|(a,b) \in U)$$

As each a_i and b_i are observed the terms in the product are of the form m/u or $(1-m)/(1-u)$ depending upon whether the a_i and b_i are equal.

Generally each variable is assumed to have the same m probability for all its levels, whereas the u probabilities can be different for different variable levels. m probabilities are related to data quality – we generally expect records relating to the same population units to match on key variables. u probabilities can be estimated from data, perhaps external data and / or the data contained in A and B. The prior odds can be estimated based on the process that has generated A and B, or from the data in A and B via, for example, maximum likelihood.

Felligi and Sunter take the logs (to the base 2) of the likelihood ratios and sum them to generate match weights. Thresholds on the match weights are used to allocate possible matches to one of three sets,

A_1 – a set of correct links

A_2 – a set of uncertain links

A_3 – a set of incorrect links

Pairs of records allocated to A_2 are subjected to clerical review, they are manually inspected and subsequently allocated to either A_1 or A_3. It is clear that these thresholds can be mapped to thresholds on the product of the likelihood ratios, or on the posterior odds via consideration of the prior odds. They can also be mapped to posterior probabilities (of a correct link) via the posterior odds.

3.1 Expectation-Maximization (EM)

EM approaches allow maximum likelihood estimation in situations where the likelihood function is difficult to maximize analytically (Dempster et al., 1977). We choose an alternative likelihood function that can be maximised analytically if we observe missing data. We assume some plausible initial values for the missing data, then iteratively generate maximum likelihood estimates for model parameters, followed by the generation of new expected values for the missing data, until convergence. EM has been used in record linkage to simultaneously estimate m and u probabilities as well as the marginal probability of a correct match (Jaro, 1989). The details are presented below.

We have $i = 1, \ldots, n$ attributes and $j = 1, \ldots, N$ record pairs. We have a set M of correct matches and a set U of incorrect matches. $\gamma_i^j = 0$ if attribute i differs for record pair j, and $\gamma_i^j = 1$ if attribute i matches for record pair j.

$$m_i = Pr(\gamma_i^j = 1 | r_j \in M)$$

$$u_i = Pr(\gamma_i^j = 1 | r_j \in U)$$

$$p = \frac{|M|}{|M \cup U|}$$

$$Pr(\gamma^j | M) = \prod_{i=1}^{n} m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j}$$

$$Pr(\gamma^j | U) = \prod_{i=1}^{n} u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j}$$

The last two equations reflect the naïve Bayes assumption – the probability of an 'agreement pattern' (given M, U) over a set of variables is the product of the probabilities of agreement for each variable (given M, U).

We want to estimate the unknown parameters $\Phi = (m, u, p)$. We have,

$$Pr(\gamma^j) = Pr(\gamma^j | M) Pr(M) + Pr(\gamma^j | U) Pr(U)$$

$$Pr(\gamma^j) = p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j} + (1 - p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j}$$

and the likelihood function is,

$$f(\gamma | \Phi) = \prod_{j=1}^{N} (p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j} + (1 - p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j})$$

Let x be the complete data vector equal to $\langle \gamma, g \rangle$, where g_j = (1,0) iff the jth record $r_j \in M$ and g_j = (0,1) iff $r_j \in U$. If we introduce this vector into the likelihood we get the likelihood for the complete data.

$$f(x | \Phi) = \prod_{j=1}^{N} g_j \cdot (p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j}, (1 - p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j})^T$$

If we take logs,

$$\ln(f(x|\Phi)) = \sum_{j=1}^{N} g_j \cdot \left(\ln(p) + \sum_{i=1}^{n} \ln(m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j}), \ln(1 - p) + \sum_{i=1}^{n} \ln(u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j}) \right)^T$$

$$\ln(f(x|\Phi)) = \sum_{j=1}^{N} g_j \cdot \left(\sum_{i=1}^{n} \ln(m_i^{\gamma_i^j} (1 - m_i)^{1-\gamma_i^j}), \sum_{i=1}^{n} \ln(u_i^{\gamma_i^j} (1 - u_i)^{1-\gamma_i^j}) \right)^T + \sum_{j=1}^{N} g_j \cdot (\ln(p), \ln(1 - p))^T .$$

For the expectation step we calculate the expectation of the missing data, g_j given starting values for the m_i, u_i and p. The expectations for the g_j are simply the probabilities that r_j is a member of M and U.

$$g_m(\gamma^j) = \frac{p\prod_{i=1}^n m_i^{\gamma_i^j}(1-m_i)^{1-\gamma_i^j}}{p\prod_{i=1}^n m_i^{\gamma_i^j}(1-m_i)^{1-\gamma_i^j} + (1-p)\prod_{i=1}^n u_i^{\gamma_i^j}(1-u_i)^{1-\gamma_i^j}}$$

$$g_u(\gamma^j) = \frac{(1-p)\prod_{i=1}^n u_i^{\gamma_i^j}(1-u_i)^{1-\gamma_i^j}}{p\prod_{i=1}^n m_i^{\gamma_i^j}(1-m_i)^{1-\gamma_i^j} + (1-p)\prod_{i=1}^n u_i^{\gamma_i^j}(1-u_i)^{1-\gamma_i^j}}$$

So each g_j is equal to $(g_m(\gamma^j), g_u(\gamma^j))$.

Now we find the maximum likelihood parameters for the m_i, u_i and p. given the complete data - the γ_i^j and the g_j. For example, for p,

$$\frac{d}{dp}\ln(f(x|\Phi)) = \sum_{j=1}^N g_m(\gamma^j).\frac{1}{p} - \sum_{j=1}^N g_u(\gamma^j).\frac{1}{(1-p)}$$

Equating to zero,

$$\sum_{j=1}^N g_m(\gamma^j).\frac{1}{p} = \sum_{j=1}^N g_u(\gamma^j).\frac{1}{(1-p)}$$

and substituting,

$$\sum_{j=1}^N g_u(\gamma^j) = N - \sum_{j=1}^N g_m(\gamma^j)$$

and rearranging, we get,

$$p = \frac{\sum_{j=1}^N g_m(\gamma^j)}{N}.$$

For the m_i,

$$\frac{d}{dm_i}\ln(f(x|\Phi)) = \sum_{j=1}^N g_m(\gamma^j).\frac{d}{dm_i}\ln(m_i^{\gamma_i^j}(1-m_i)^{1-\gamma_i^j})$$

$$\frac{d}{dm_i}\ln(f(x|\Phi)) = \sum_{j=1}^N g_m(\gamma^j).(\frac{\gamma_i^j}{m_i} - \frac{1-\gamma_i^j}{1-m_i}).$$

Equating to zero,

$$\sum_{j=1}^N g_m(\gamma^j).\frac{\gamma_i^j}{m_i} = \sum_{j=1}^N g_m(\gamma^j).\frac{1-\gamma_i^j}{1-m_i}$$

and rearranging,

$$m_i = \frac{\sum_{j=1}^N \gamma_i^j \cdot g_m(\gamma^j)}{\sum_{j=1}^N g_m(\gamma^j)}.$$

Similarly for the u_i,

$$u_i = \frac{\sum_{j=1}^N \gamma_i^j \cdot g_u(\gamma^j)}{\sum_{j=1}^N g_u(\gamma^j)}.$$

These estimates are used as fixed parameters for the next expectation step. This generates new $(g_m(\gamma^j), g_u(\gamma^j))$ which are used to generate the subsequent estimates for m_i, u_i and p, and so on, until convergence.

3.2 An alternative EM approach

The approach can be extended to variables with more than two levels. Winkler (1992) noted that binary comparison could perform poorly in distinguishing between distinct individuals in the same household. Winkler adopted a 3-class procedure, allowing non-matches to be partitioned into distinct individuals within a household and distinct individuals in different households – the complement of the union of these two classes being the same individuals in the same household, M.

Consider an extension to K categories, $k = 1, \ldots, K$, where the categories are similarity score intervals.

$$m_{i,k} = Pr(\gamma^j_{i,k} = 1 | r_j \in M)$$

$$u_{i,k} = Pr(\gamma^j_{i,k} = 1 | r_j \in U)$$

$$p = \frac{|M|}{|M \cup U|}$$

$$Pr(\gamma^j|M) = \prod_{i=1}^{n}\prod_{k=1}^{K} m_{i,k}^{\gamma^j_{i,k}}$$

$$Pr(\gamma^j|U) = \prod_{i=1}^{n}\prod_{k=1}^{K} u_{i,k}^{\gamma^j_{i,k}}$$

We define g as before. The complete data log likelihood is,

$$\ln(f(x|\Phi)) = \sum_{j=1}^{N} g_j \cdot \left(\sum_{i=1}^{n} \ln(\prod_{k=1}^{K} m_{i,k}^{\gamma^j_{i,k}}), \sum_{i=1}^{n} \ln(\prod_{k=1}^{K} u_{i,k}^{\gamma^j_{i,k}})\right)^T + \sum_{j=1}^{N} g_j \cdot (\ln(p), \ln(1-p))^T .$$

Estimation of p is clearly unaffected by the extension to K categories as the derivative (with respect to p) is identical to the previous case. For the $m_{i,k}$ and $u_{i,k}$ we need the maximum likelihood parameters of a multinomial,

$$m_{i,k} = \frac{\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_m(\gamma^j)}{\sum_{j=1}^{N} g_m(\gamma^j)}$$

$$u_{i,k} = \frac{\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_u(\gamma^j)}{\sum_{j=1}^{N} g_u(\gamma^j)}.$$

This agrees with the results in Appendix A of Yancey (2002), illustrating the procedure for the 3-class approach.

It should be noted that when $\gamma^j_{i,k} = 0$ for all j, then the estimated $m_{i,k}$ and $u_{i,k}$ will equal 0. In this case the product terms in the complete data log likelihood will contain terms 0^0. In this context the interpretation must be that $0^0 = 1$- not observing the impossible has probability 1. This has two implications.

For a given variable we can have categories that can never be observed, and the above theory still holds. In other words, distinct variables can have distinct numbers of levels.

We have a natural way of handling missing values. If the jth record pair contains a missing value (for either record) for the ith variable, then $\gamma^j_{i,k} = 0$ for all k. With missing data we simply need to adjust the maximum likelihood estimation of the multinomial parameters,

$$m_{i,k} = \frac{\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_m(\gamma^j)}{\sum_{k=1}^{K}\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_m(\gamma^j)}$$

$$u_{i,k} = \frac{\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_u(\gamma^j)}{\sum_{k=1}^{K}\sum_{j=1}^{N} \gamma^j_{i,k} \cdot g_u(\gamma^j)}.$$

This simply represents a re-normalization of the $m_{i,k}$ and $u_{i,k}$ so that for each variable they sum to 1 across the categories.

3.3 Similarity scores

The issue is how to best incorporate similarity scores into the record linkage process. EpiLink (Contiero et al., 2005) uses similarity scores for matching. Attributes are given weights, and the "S value" for a pair of records is the weighted sum of the similarity scores for the key variables,

$$S(a,b) = \frac{\sum_i w_i s(a_i,b_i)}{\sum_i w_i}.$$

The following weight function is recommended,

$$w_i = \log_2\left(\frac{1-e_i}{f_i}\right)$$

where e_i is the error rate for attribute i and f_i is (what the authors call) the "average frequency of values in the field". It appears to be the average proportion of values, which just works out as the reciprocal of the number of distinct values for a field.

(Note: Contiero et al., 2005, give the equation $w_i = \log_2(1-e_i)/f_i$ but their examples imply that the above equation is actually the correct one.)

These weights are broadly the same as the match weights in Fellegi-Sunter. This suggests that EM could be used to generate the EpiLink weights.

Other approaches tend to generate likelihood ratios (or match weights) using the traditional (binary comparison) Fellegi-Sunter framework, then perform interpolation using similarity scores to produce a new match weight (or adjusted likelihood ratio). For instance, a basic approach for interpolating on the likelihood ratios gives,

$$\frac{Pr((a_i,b_i)|(a,b)\in M)}{Pr((a_i,b_i)|(a,b)\in U)} = s(a_i,b_i)\frac{m_i}{u_i} + (1-s(a_i,b_i))\frac{1-m_i}{1-u_i}.$$

Interpolating on the $\log_2$ likelihood ratios (match weights) is equivalent to the following,

$$\frac{Pr((a_i,b_i)|(a,b)\in M)}{Pr((a_i,b_i)|(a,b)\in U)} = \left(\frac{m_i}{u_i}\right)^{s(a_i,b_i)}\left(\frac{1-m_i}{1-u_i}\right)^{1-s(a_i,b_i)}.$$

Winkler (1990) uses the Jaro string comparator and an additional parameter to adjust the weighting,

$$\frac{Pr((a_i,b_i)|(a,b)\in M)}{Pr((a_i,b_i)|(a,b)\in U)} = max\left[\left(\frac{m_i}{u_i}\right)^{1-(1-s(a_i,b_i)c)}\left(\frac{1-m_i}{1-u_i}\right)^{1-s(a_i,b_i)c}, \frac{1-m_i}{1-u_i}\right]$$

setting $c = 9/2$.

He also outlines an approach based on splitting the similarity score into intervals, estimating the likelihood ratio for each interval $(k_j, k_{j+1}]$(using other datasets),

$$\frac{Pr(s(a_i,b_i)\in(k_j,k_{j+1}]|(a,b)\in M)}{Pr(s(a_i,b_i)\in(k_j,k_{j+1}]|(a,b)\in U)}$$

then using the interpolation approach above. The value of c for each interval is chosen so that the interpolated likelihood ratios approximate those fitted from the external data.

The traditional binary comparison is equivalent to two intervals, the degenerate interval $(1,1]$ and the non-degenerate interval $[0,1)$. Clearly the numerator and denominator of the likelihood ratio for the interval $(1,1]$ are going to be less than, or equal to, the equivalent terms for a wider interval containing 1. There is no guarantee that the likelihood ratio for the wider interval will be bounded above by that for the interval $(1,1]$. Similarly, there is no guarantee that the likelihood ratios for intervals will be bounded below by that for the interval $(1,1]$. Intuitively we would expect a lower likelihood ratio for intervals containing only very low similarity scores. The interpolation approach appears to overly restrict the likelihood ratios for intervals. An alternative is simply to use the likelihood ratios for the intervals derived from external data – perhaps employing some other form of interpolation. The approach adopted here is to use similarity score intervals, and to use the multinomial EM approach to estimate the parameters.

3.4 Multinomial EM using similarity scores

It is not uncommon in statistical modelling and computer science to categorize continuous variables in order to make certain methods available or to reduce computational costs (e.g. inference in Bayesian networks). Here we use the multinomial EM approach using categorized similarity scores rather than the traditional binary string agreement variable. Thus we generate m and u probabilities for each similarity score level (for each key variable). This has a number of advantages.

Fewer assumptions are required of the similarity score – for instance, it need not be in the interval [0, 1]. We do not need to consider whether it is more appropriate to interpolate on the weights or the likelihood ratios. We do not need to consider monotone transformations of the similarity scores to improve the weighting – in effect, the fitting procedure takes care of this automatically. The resulting likelihood ratio for a pair of strings is not restricted to the range implied by the likelihood ratios under the binary comparison assumption.

Although alternative similarity scores might be more or less suitable for certain types of variable, once the score is chosen we need only consider the appropriateness of the categorization. (In terms of privacy preserving linkage using Jaccard scores, the choice of score becomes the choice of tokenization scheme.)

Consider inferences based on the binary comparison scheme with and without similarity score weighting. Inferences are identical only when similarity scores equal 0 or 1. Under the generally reasonable assumption that $\frac{m_i}{u_i} > \frac{1-m_i}{1-u_i}$ likelihood ratios (and posterior odds and posterior probabilities of a correct link) will be inflated for scores between 0 and 1. Although this can help to discriminate between correct and incorrect links, it will require adjustments to any thresholds based on theoretical considerations (see Section 3.5.3). Easily interpretable parameter estimates will require that the similarity scores are incorporated into the fitting process. That is what is proposed here. This approach is simpler than the interpolation schemes, easily implemented, and more grounded in theory.

3.5 Blocking and thresholds

For large A and B the set of all possible matches can be very large. This can lead to computational issues, so a common practice is to eliminate from consideration low probability links. Also, increasing the proportion of correct links in the pairs used for the EM approach can lead to improved estimation (Yancey, 2002). One approach is to identify key variables that are considered to be particularly reliable and to require exact matching on these variables (Jaro, 1989). This is termed standard *blocking*. However, there are alternative strategies for blocking. Similarity score can be used for blocking – assigning all possible links below a given threshold to the set of incorrect links, A_3.

For pairs of records that are not allocated to A_3 via blocking, classification is based on thresholds on the overall match weight for a pair of records (or equivalently on the likelihood ratio or posterior probability of a correct link).

So in putting together a record linkage system based on Fellegi-Sunter theory, we have two further issues that must be addressed,

1. How do we implement blocking in order to reduce computational costs?
2. Where do we set our thresholds for classification of record pairs?

These questions which will be addressed in the following sections. But first we will summarise an important aspect of Fellegi and Sunter's paper – their decision rule.

3.5.1 Fellegi-Sunter decision rule

Fellegi and Sunter (1969) specify a linkage rule as a decision function,

$$d(\gamma) = \{P(A_1|\gamma), P(A_2|\gamma), P(A_3|\gamma)\}$$

where $\gamma \in \Gamma$ and Γ is the comparison space.

The comparison space is the Cartesian product of the sets of variable levels for the key variables. So for binary comparisons and n variables the comparison space has size 2^n - there are 2^n distinct configurations of the evidence that we might see for a pair of records. Each comparison vector γ is associated with probabilities $m(\gamma)$ and $u(\gamma)$, and a likelihood ratio $m(\gamma)/u(\gamma)$. Under the naïve Bayes assumption, $m(\gamma)$ and $u(\gamma)$ are the products of the m and u probabilities associated with the vector components.

There are two types of error associated with a decision rule,

$$P(A_1|U) = \mu$$

and

$$P(A_3|M) = \lambda.$$

Fellegi and Sunter (1969) define an optimal decision rule as a rule that minimizes the probability of assignment to A_2 whilst maintaining μ and λ . They present an algorithm for generating an optimal decision rule, $L_0 = (\mu, \lambda, \gamma)$.

They define an ordering on the possible realizations of γ . γ for which $m(\gamma) = u(\gamma) = 0$ are excluded from the ordering. γ for which $m(\gamma) > 0$ and $u(\gamma) = 0$ are ordered arbitrarily. These are then followed by the γ in decreasing order of $m(\gamma)/u(\gamma)$. The ordered γ are subscripted $i = \{1, \ldots, N_\Gamma\}$. Let $m_i(\gamma)$ and $u_i(\gamma)$ be denoted m_i and u_i respectively.

Choose n and n' such that,

$$\sum_{i=1}^{n-1} u_i < \mu \leq \sum_{i}^{n} u_i$$

$$\sum_{i=n'}^{N_\Gamma} m_i \geq \lambda \sum_{i=n'+1}^{N_\Gamma} m_i.$$

Then the optimal decision rule is,

$$d(\gamma_i) = \begin{cases} (1,0,0) & i \leq n-1 \\ (P_\mu, 1-P_\mu, 0) & i = n \\ (0,1,0) & n < i \leq n'-1 \\ (0, 1-P_\lambda, P_\lambda) & i = n' \\ (0,0,1) & i \geq n'+1 \end{cases}$$

where.

$$u_n P_\mu = \mu - \sum_{i=1}^{n-1} u_i$$

$$m_{n'} P_\lambda = \lambda - \sum_{i=n'+1}^{N_\Gamma} m_i.$$

Consideration of the Fellegi-Sunter decision rule makes it clear how blocking will affect the error rates. Blocking will allocate record pairs with certain comparison vectors to A_3. This places a lower bound on λ. We can use the decision rule algorithm to calculate this lower bound given sets of m and u probabilities. In effect we can use the same algorithm to investigate the error rates associated with various blocking schemes by changing the ordering of the comparison vectors. We simply take the comparison vectors that result in allocation to A_3 (due to blocking) and place them at the end of the list. The sum of the m_i associated with these comparison vectors is the lower bound on λ because these comparison vectors contribute to λ regardless of any subsequently chosen decision rule. Similarly, 1 minus the sum of the u_i associated with these comparison vectors is an upper bound on μ.

In the appendices of their 1969 paper Fellegi and Sunter present a plot that is useful for illustrating the relationships between error rates and allocation thresholds. It is a piecewise linear function constructed from the ordered comparison vectors. Each straight line section between points $(x_a,\ y_a)$ and $(x_b,\ y_b)$, $x_a < x_b$, corresponds to a comparison vector γ such that $m(\gamma) = y_a - y_b$ and $u(\gamma) = x_b - x_a$. Thus it is a plot of λ against μ for decision rules with empty A_2. Admissible values for λ and μ with non-empty A_2 are given by points to the left of / below the piecewise linear function.

Figure 2 shows a plot of λ against μ for two sets of records with m and u probabilities estimated using an Expectation-Maximization algorithm. The data are those used for the experiments shown in Section 4. The effect of blocking on one of the variables is shown in Figure 3. Blocking generates a lower bound on λ of around 0.0072 and an upper bound on μ of around 0.1008.

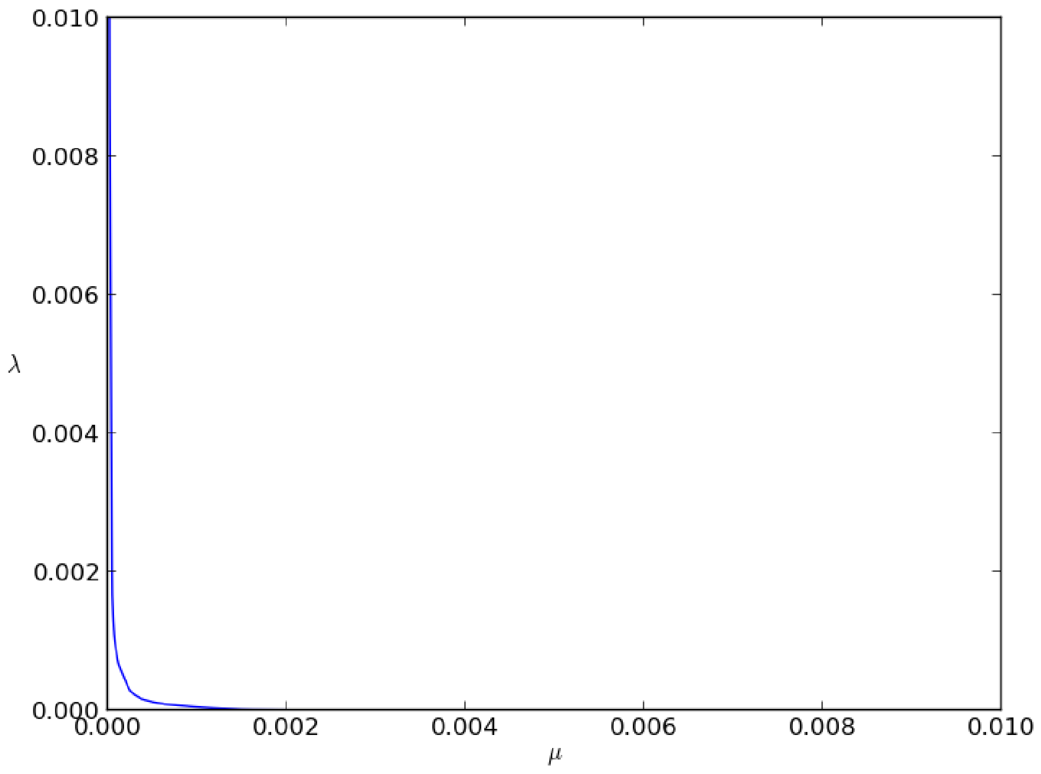

Figure 2: Admissible region for error rates without blocking

With no opportunity for clerical review we are limited to decision rules that lie on the piecewise linear function. That is, we have a single threshold used for allocation to A_1 or A_3. A threshold can be expressed in terms of the likelihood ratio for a comparison vector. It partitions the comparison vectors into those that result in allocation to A_1 and those that result in allocation to A_3. Thus the plot relates both λ and μ , and thresholds via the gradient of the straight line sections of the piecewise linear function. We can, in principle, take a threshold or an error rate and calculate the other two quantities, with or without blocking.

Appendix A contains the implementation details for a revised decision rule algorithm.

3.5.2 Blocking

Under blocking we can simply remove the 'non match levels' from the relevant sub-lists before generation in decreasing order of $m(\gamma)/u(\gamma)$. This will be sufficient for finding a threshold for any admissible μ (or vice versa). Then we can repeat the generation of comparison vectors (in an arbitrary order if we want) with only the non match levels for the relevant sub-lists. The sum of the m_i for this second iteration is a lower bound on λ . A more efficient approach to calculate this lower bound is to simply calculate the product over the match levels and subtract it from 1. We then iterate over the 'match' comparison vectors in increasing order to find our actual threshold for a given value of λ (or vice versa).

It should be clear from the above discussion that blocking need not be on exact matches. It is possible to engineer a tokenization and categorization scheme so that we have a similarity score level that corresponds to an exact match. But we might have a 'match level' that corresponds to a similarity score in e.g. the interval (0.8, 1]. Thus we would be blocking on a similarity score threshold of 0.8. The scheme can easily be adapted to include more than one 'match level' for blocking, whilst retaining the finer grained information in the (categorized) similarity scores for matching purposes. For instance, we might have 2 categories (0.8, 0.999] and (0.999, 1] which are treated as 'match levels' for blocking purposes, but which have distinct m and u probabilities used for generating likelihood ratios. Thus we have the potential to investigate both alternative sets of variables for blocking, and alternative similarity score thresholds.

The above provides a means of assessing blocking schemes. But it is based on having estimated m and u

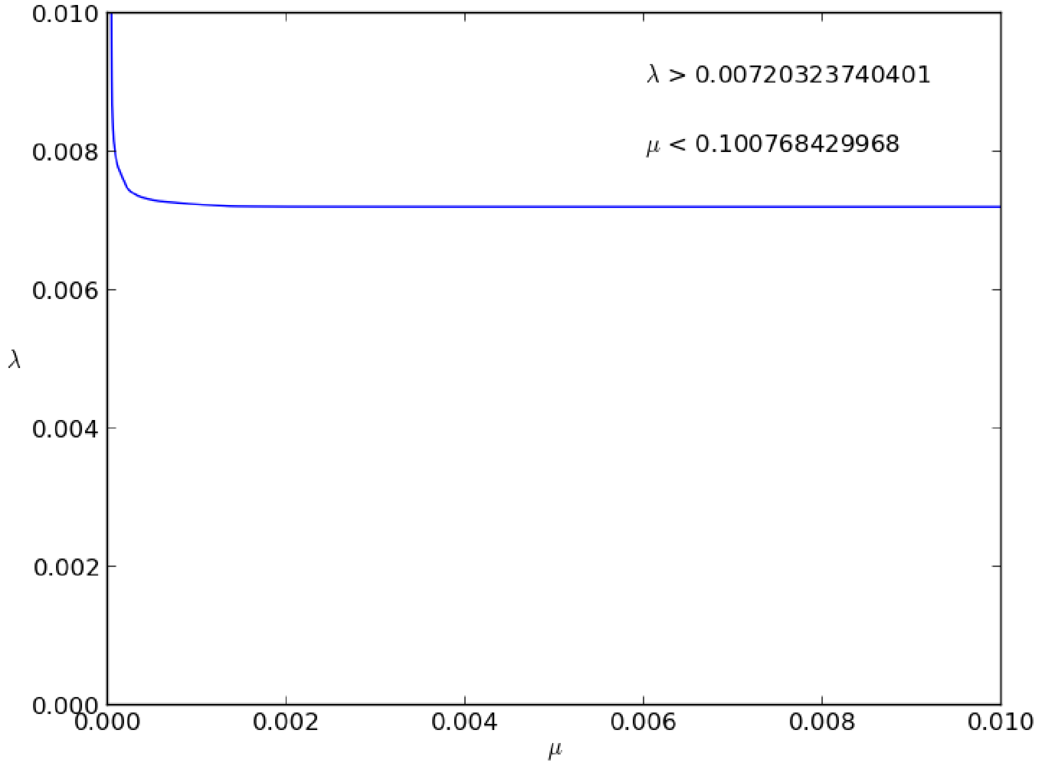

Figure 3: Admissible region for error rates with blocking

probabilities, and blocking is designed to reduce the number of record pairs under consideration so as to reduce the computational cost of their generation. In practice we can generate a random sample of record pairs and use those for generating m and u probabilities and assessing blocking schemes. Once the blocking scheme is decided, then all the relevant (non blocked) record pairs can used for classification.

Clearly, even in a privacy preserving context, we can still use standard blocking based on exact matching of variable values (Jaro, 1989). Given a suitable tokenization scheme there will be a 1 to 1 correspondence from pairs of exact matches to similarity scores of 1.

Some approaches are based on exact matching of tokens. For instance, the sorted-neighbourhood method (Hernández and Stolfo, 1998) performs multiple passes over the data looking for matches (or close matches) on keys constructed from tokens. If Alice and Bob were asked to generate such tokens from the data, then they could be anonymized using concatenated 1-bit hashing. The resulting fields would be used solely for blocking. The sorted neighbourhood method could be applied directly using anonymized tokens.

Other methods are based on distance measures, and concatenated 1-bit hashes can be used directly. Canopy clustering (McCallum et al., 2000) is a two stage clustering approach using a pair of thresholds. A cheaply evaluated distance measure is used to divide keys into overlapping subsets termed *canopies*. A more expensively evaluated measure is used to identify similar pairs of keys within a canopy. The following pseudoPython code illustrates the approach.

```
def get_canopies(keys, t1, t2):
    # keys is a set
    # t1 and t2 are numeric values
    canopies = []
    while keys is not empty:
        c = set()
        k = keys.pop()
```

```
            c.add(k)
            for k2 in keys:
                if dist1(k,k2) < t1:
                    c.add(k2)
                    if dist2(k,k2) < t2:
                        keys.remove(k2)
            canopies.append(c)
        return canopies
```

dist1 and *dist2* might be the same distance measure, in which case t1 > t2. Pairs of keys that are not members of a common canopy are considered to be dissimilar. In principle, for blocking, we could use a multi-pass approach generating sets of canopies for individual variables, or we could use a single pass with a composite key. Concatenated 1-bit hashes lend themselves to this approach, as distance measures can be generated directly from the hashes. Composite key generation simply requires a number of hashes to be concatenated to form a single hash. (In this case the estimated Jaccard score generated is the estimated mean Jaccard score over the component concatenated 1-bit hashes – assuming the components are of the same length.) If concatenation produces hashes of excessive length, then they can be compressed by removing the bits at randomly selected bit positions. In fact, this form of compression can be the basis of the similarity score evaluations. We might use a compressed hash for the *dist1* evaluation and a longer hash for the *dist2* evaluation, and thresholds such that t1 > t2. Efficiency largely depends on the cost of identification of the keys/hashes that are within a given (estimated) similarity of a target key/hash. The *for* loop in the above code suggests that each key must be considered. The following demonstrates that this is not necessarily the case, depending on the data structure holding the keys/hashes.

Bachteler et al. (2013) generate a single Bloom filter for each configuration of the evidence by using the same approach as described in Schnell et al. (2009) and using distinct sets of hash functions for each key variable. They use multibit trees (Kristensen et al., 2010) to identify comparison vectors that have an (estimated) Jaccard score above a given threshold.

A multibit tree has a single source node, v. All non sink nodes have two outgoing edges – labelled 0 and 1. Thus the labels on the edges along any path $\{v, \ldots, w\}$ from the source node to a sink node can represent a bitstring. Each sink node w is associated with the bitstring corresponding to the edges on the path $\{v, \ldots, w\}$, which in turn is associated with a list of corresponding record IDs. A multibit tree can be constructed from the Bloom filters associated with the comparison vectors in dataset A, then searched for similar bitstrings to each of the Bloom filters associated with the comparison vectors in dataset B. Bounds on the (estimated) Jaccard scores are calculated as search proceeds, allowing the search to be pruned. (See Appendix B for more details and discussion.)

As they use a single Bloom filter for each configuration of the evidence they are, in fact, blocking on the average (estimated) Jaccard score over the key variables.

The Jaccard score is estimated from two Bloom filters by,

$$\widehat{J}_{A,B} = \frac{|A \wedge B|}{|A \vee B|},$$

the number of set bits in the bitwise AND of A and B divided by the number of set bits in the bitwise OR of A and B.

It is clear that multibit trees might offer a means of reducing the costs associated with the *for* loop in the canopy clustering approach. Bit sampling (e.g. Durham, 2012) could be used to generate compressed Bloom filters for the evaluation of *dist1* whilst, within each canopy, a multibit tree constructed from the added Bloom filters could be used for the evaluation of *dist2* based on the full complement of bits.

It is also possible to use multibit trees with concatenated 1-bit hashes. In fact, there are some advantages to the use of concatenated 1-bit hashes, mainly due to the simplicity of calculating bounds when compared to the bounds calculations for Bloom filters (see Kristensen et al., 2010, for details).

Appendix B contains the implementation details of a blocking approach based on tries, which has some similarity to Kristensen et al.'s multibit tree approach. Details of the handling of missing values are also presented.

3.5.3 Thresholds

We have two error rates associated with a decision rule,

$$P(A_1|U) = \mu$$

and

$$P(A_3|M) = \lambda.$$

The decision rule algorithm can be used to identify thresholds that correspond to given error rates. In a privacy preserving record linkage context (no clerical review) we have a single threshold. We can choose this to correspond to a given value for either error rate. However, it is clear from the plots such as Figure 2 that we could choose a threshold for which one of the rates could be reduced substantially for only a small increase in the other error rate. It would seem best to try to avoid such situations, perhaps visually inspecting a plot to choose an appropriate threshold.

The marginal probabilities of false positives and false negatives are $\mu(1-p)$ and λp respectively. If we are concerned about these error rates, rather than the conditional error rates above, then we need to make appropriate adjustments.

We might plot λp against $\mu(1-p)$ (simply a rescaling of the axes) and look for a reasonable threshold.

More generally, we might want to choose a threshold to minimize a cost function,

$$z = C_{MU}P(A_1|U)P(U) + C_{UU}P(A_3|U)P(U) + C_{UM}P(A_3|M)P(M) + C_{MM}P(A_1|M)P(M)$$

where C_{XY} is the cost of allocating a pair to class X when the correct classification is class Y.

$$z = C_{MU}\mu(1-p) + C_{UU}(1-\mu)(1-p) + C_{UM}\lambda p + C_{MM}(1-\lambda)p$$

$$z = (C_{MU} - C_{UU})\mu(1-p) + C_{UU}(1-p) + (C_{UM} - C_{MM})\lambda p + C_{MM}p$$

This can be rearranged,

$$\lambda = -\frac{(C_{MU} - C_{UU})(1-p)}{(C_{UM} - C_{MM})p}\mu + \frac{z - C_{UU}(1-p) - C_{MM}p}{(C_{UM} - C_{MM})p}$$

showing that, expressed in terms of λ as a function of μ , z defines a family of functions with slope,

$$-\frac{(C_{MU} - C_{UU})(1-p)}{(C_{UM} - C_{MM})p}.$$

The feasible solutions occur at the intersections of the straight-line functions that make up the curves in e.g. Figure 2. Minimization of z is equivalent to the minimization of the intercept term above.

If the gradient above is negative, then we have a solution at the point of intersection of straight-line sections such that

$$\frac{m_j}{u_j} \geq \frac{C_{MU} - C_{UU}}{C_{UM} - C_{MM}} \geq \frac{m_{j+1}}{u_{j+1}}.$$

Convexity of the curve ensures that we have exactly one solution, unless we have equality above, in which case we have two solutions corresponding to the endpoints of a single straight-line section. In either case the threshold for the likelihood ratio is,

$$\frac{(C_{MU} - C_{UU})(1-p)}{(C_{UM} - C_{MM})p}.$$

If $C_{MU} > C_{UU}$ and $C_{UM} > C_{MM}$, then we allocate comparison vectors with likelihood ratios above the threshold to A_1, and comparison vectors with likelihood ratios below the threshold to A_3. (Comparison vectors equal to the threshold can be allocated arbitrarily, or we can allocate so as to favour either μ or λ .)

Less usual cases imply alternative strategies, but the rules for allocation can be derived from examination of z.

It might be that we want to minimize z subject to an upper limit on one of the error rates. To establish whether a constraint on an error rate is satisfied we have to conduct a search. Minimizing z tells us nothing about our actual error rates. The effect of a constraint is to reduce the set of feasible solutions. But convexity implies that the solution will either be the solution to the unconstrained problem, or the threshold that satisfies the error rate limit exactly. We can conduct a search until we either generate the threshold associated with the error rate limit or the error rate associated with the unconstrained solution. At this point we can decide which threshold is the solution to the constrained problem. The above extends naturally to constraints on both error rates.

In terms of the posterior probability of a match the threshold (for the unconstrained problem) is,

$$\frac{C_{MU} - C_{UU}}{C_{UM} - C_{MM} + C_{MU} - C_{UU}}.$$

In particular, if $C_{MU} = C_{UM}$ and $C_{UU} = C_{MM} = 0$ then the threshold is a posterior probability of 0.5 which minimizes the sum of the unconditional error rates (e.g. Elmagarmid et al., 2007).

If $C_{MU}/P(U) = C_{MM}/P(M)$ and $C_{UU} = C_{MM} = 0$ then the threshold is a posterior probability of $P(M)$. In other words, allocating on the basis of the preponderance of the evidence (likelihood ratio of 1) minimizes the sum of the conditional error rates.

Minimizing a linear function of μ and λ requires no special treatment at the threshold value - unlike when maintaining specified values for μ and λ .

So we can exploit the decision rule algorithm to identify thresholds that relate to specific error rates, or we can employ thresholds based on theory.

4 Record linkage experiments

A subset of 1000 records (File A) were taken from a database containing census data and randomly generated first names. A second file (File B) was generated by perturbing File A according to the following scheme.

Gender - integer coded {1,2}:

> Swap code with probability 1/80. Double up a character with probability 1/40. Delete value with probability 1/40.

Year of birth:

> Transpose 2 adjacent characters with probability 1/40. Delete value with probability 1/60.

Month of birth – integer code 1-12:

> Delete a character with probability 1/40.

First name:

> Transpose 2 adjacent characters with probability 1/20. Delete a character with probability 1/60. Double up a character with probability 1/60. Substitute character with a keyboard adjacent character with probability 1/60. Delete value with probability 1/60.

Perturbations were applied in the orders given above. Characters (for e.g. deletion) / character pairs (for e.g. transposition) were selected with equal probability. In total 4 perturbed datasets were generated, one using the above scheme and 3 further datasets using the same perturbations but with the probabilities doubled, trebled and quadrupled. In each case a random sample of 700 records from File A and a random sample of 400 records from the perturbed file were used for matching. Steps were taken to correctly handle missing values (under the 'missing at random' assumption). Matching experiments compared the following seven approaches.

4.1 Linkage approaches

Binary EM:

> This was the standard EM approach based on exact matching of strings. Similarity scores were not used.

LR weighted:

> This used the outputs of Binary EM weighting the likelihood ratios as detailed in Section 3.3.

Log LR weighted:

> This used the outputs of Binary EM weighting the $\log_2$ likelihood ratios as detailed in Section 3.3.

EM (8):

> The multinomial EM approach was used with 8 categories with (inclusive) upper bounds [0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 0.999, 1]. The similarity score was the Jaccard score. Strings were padded with a leading and a trailing underscore - the bigrams of the resulting strings were taken as the token sets.

EM (15):

> The multinomial EM approach was used with 15 categories with (inclusive) upper bounds [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.925, 0.95, 0.975, 0.999, 1]. Jaccard scores were generated as for EM (8).

Jaro (Jaro, 1976):

> This used the multinomial EM approach. The Jaro similarity measure was used, along with the 8 level categorization given above.

Jaro-Winkler (Winkler, 1990) :

> This used the multinomial EM approach. The Jaro-Winkler similarity measure was used, along with the 8 level categorization given above.

4.2 Results

Correct links were identified and used to construct precision-recall plots. Possible links are allocated to a set of positive matches if they are above a given threshold (posterior probability of a correct link), otherwise they are allocated to a set of incorrect matches. For any given threshold we will have a number of false positives *fp*, and a number of false negatives *fn*. Similarly we will have number of true positives *tp*, and a number of true negatives *tn*.

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

A plot of precision against recall for a large range of thresholds (one threshold for each distinct precision, recall pair) allows the comparison of record linkage approaches. Good approaches will produce curves in the upper right of the plot.

As expected, methods generally perform better with lower levels of perturbation. There is a relatively consistent picture across all four plots. Binary EM without weighting performs the worst. Similarity score weighting of the binary EM outputs provides better performance, with weighting of the log likelihood ratios performing better. These performances are surpassed by all the schemes based on multinomial EM.

There is no clear difference in performance between the 8 category and 15 category Jaccard score schemes. The Jaro and Jaro-Winkler schemes provide the best performance, although it is important to remember that these scores are calculated from underlying strings, and cannot be used in a privacy preserving context.

Classification tables were generated for the threshold of a posterior probability of 0.5. Theory implies that this threshold should minimize the sum of the false positives and the false negatives. Error rates generally reflected the precision recall plots, with lowest rates for the Jaro and Jaro-Winkler approaches. The one

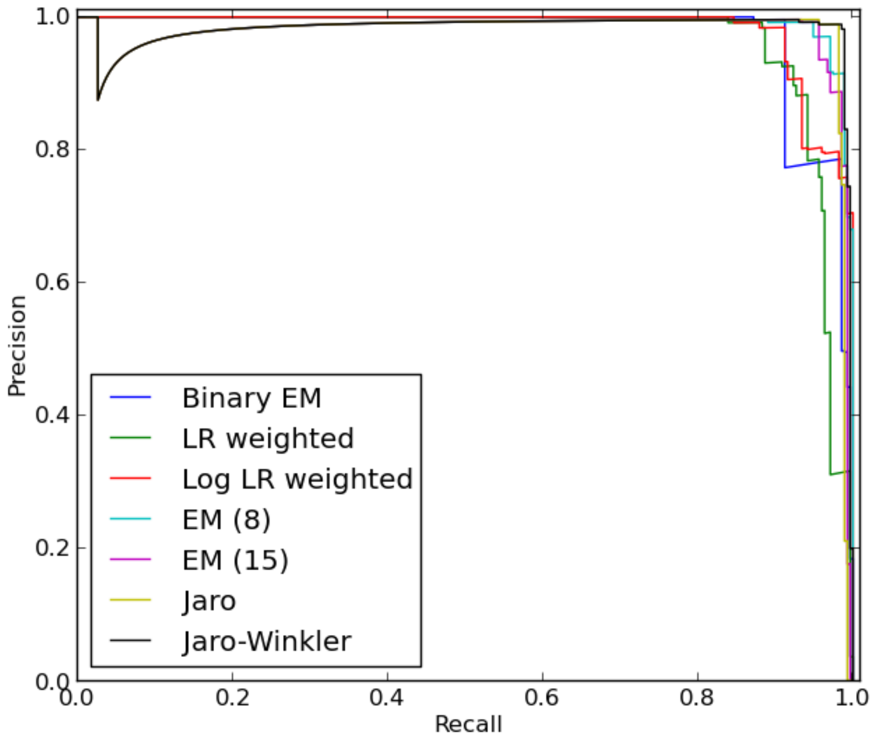

Figure 4: Precision recall graph with factor 1

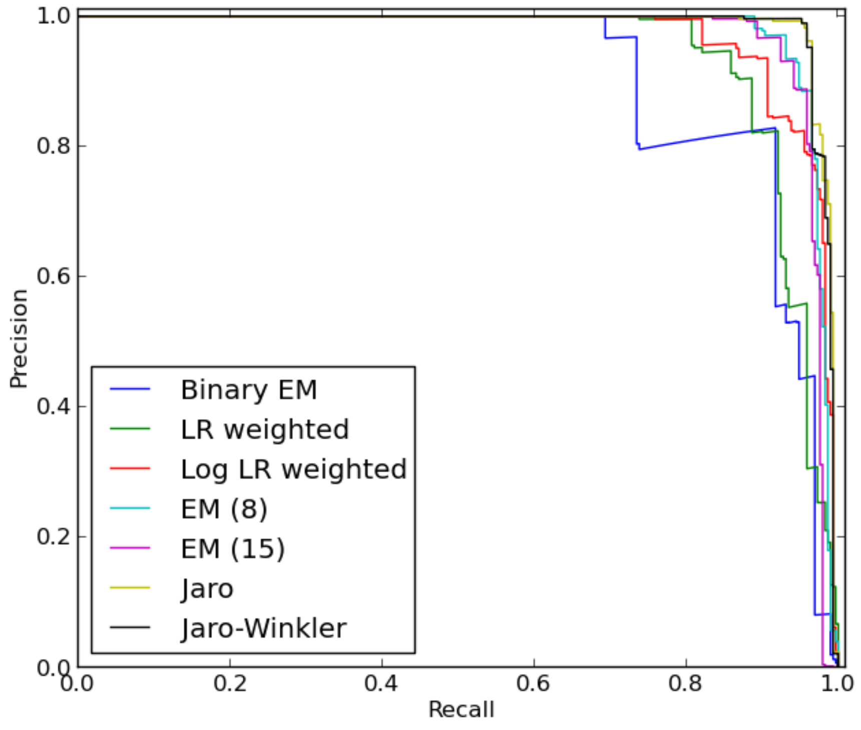

Figure 5: Precision recall graph with factor 2

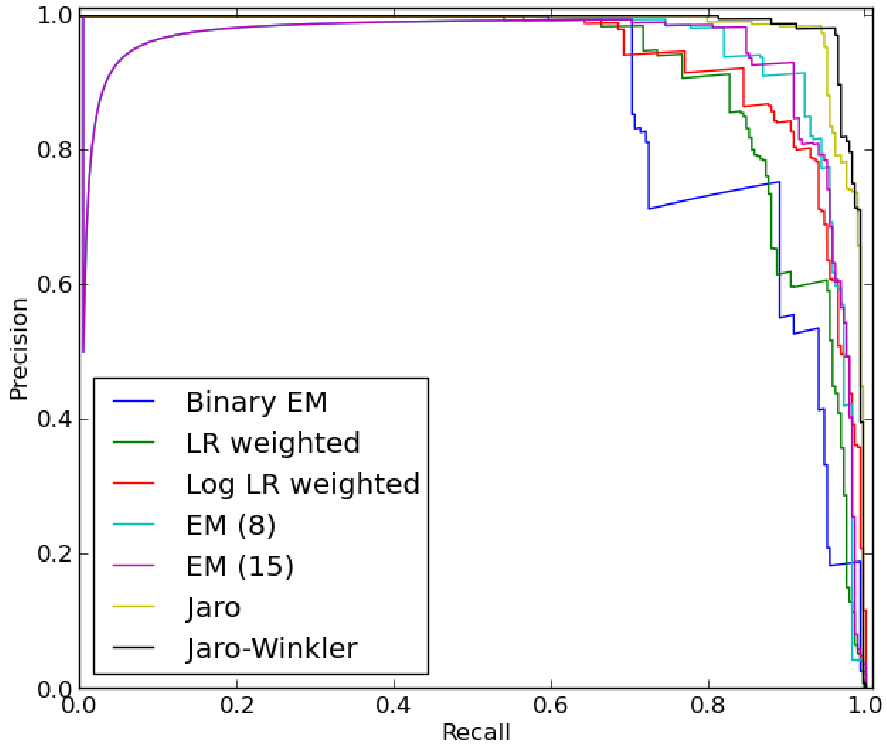

Figure 6: Precision recall graph with factor 3

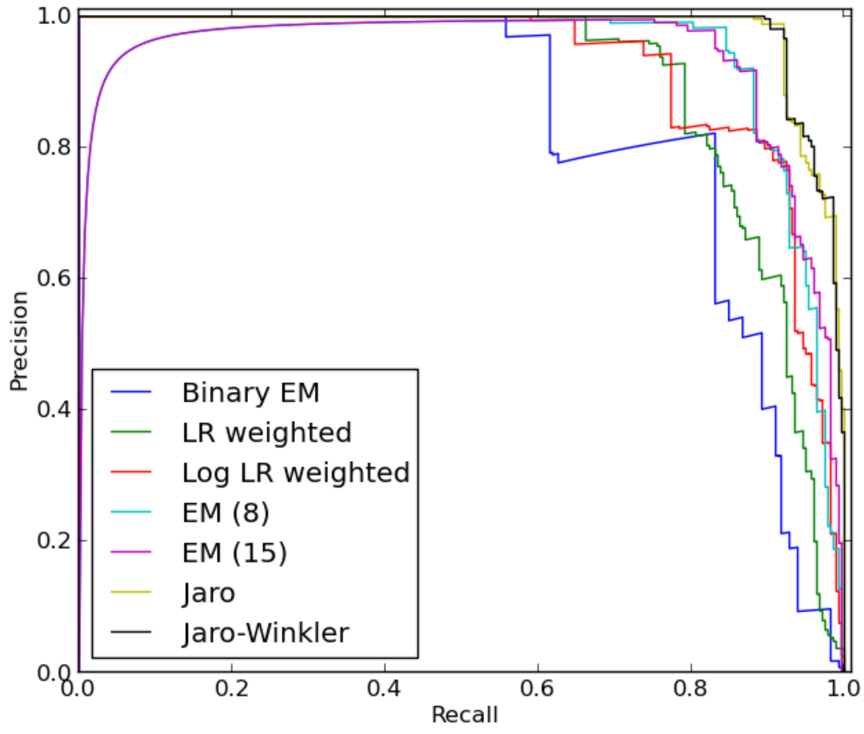

Figure 7: Precision recall graph with factor 4

exception to this was the similarity score likelihood ratio weighting scheme. In this case there were 0 false negatives and a significantly inflated number of false positives for each record set. This is clearly due to the inflation effect noted in Section 3.4. The rates appeared to be much more reasonable for the similarity score log likelihood ratio weighting scheme, but this should not be relied upon in practice. Theoretical thresholds will not generally apply under *post hoc* weighting schemes.

4.3 Computational performance

The time taken for the 8 category multinomial EM is generally around 4-16 times that of the binary EM (which took an average of around 20s for the above analyses). Costs would be greater for larger datasets. The software is implemented in Python, using a fast array library (numpy) for the relevant operations. It is doubtful that simply reimplementing the code using another programming language / library could produce much in the way of performance gains. However, it can be seen from Figure 8 that EM converges quickly, and most of the time is spent producing only small improvements to the model. Alternative convergence rules will be investigated. The largest gains are likely to come from blocking.

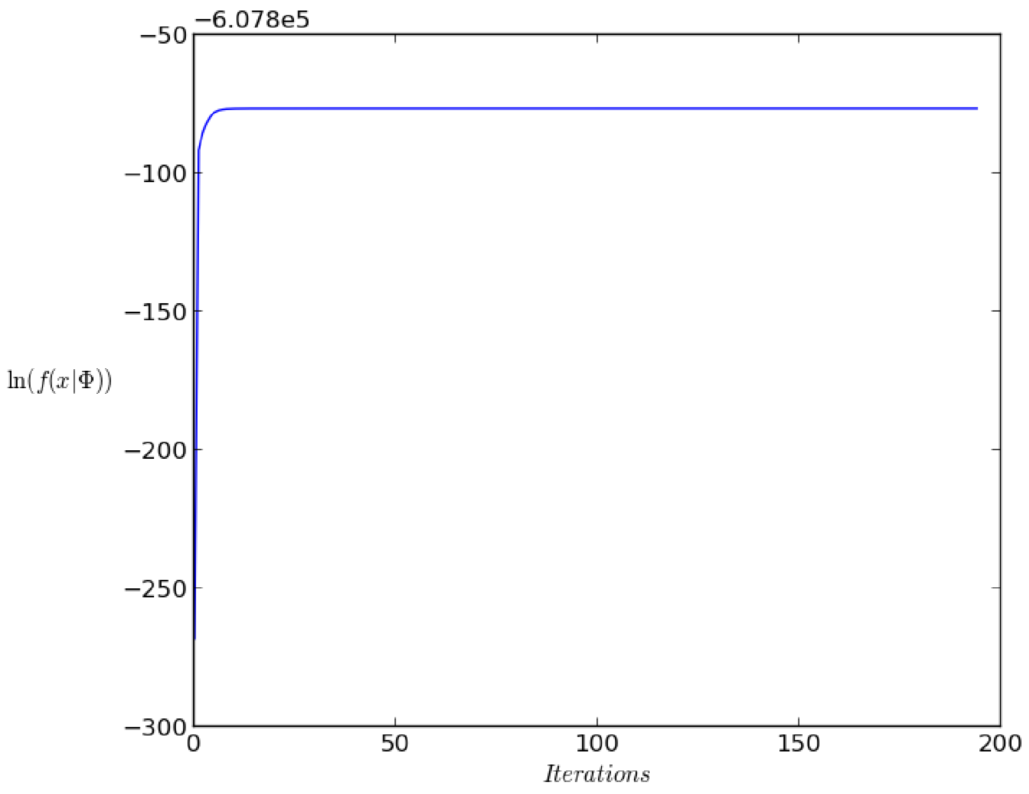

Figure 8: Convergence of EM algorithm

4.4 Discussion of experiments

Multinomial EM offers improved classification over the unweighted and weighted binary EM schemes. For the datasets investigated the Jaro and Jaro-Winkler schemes outperformed the Jaccard scheme. The perturbations were selected so as to emulate typographical errors, so this is no great surprise. For alternative sources of error, such as OCR errors, this might not be the case. An advantage of the Jaccard scheme is that tokenization can be tailored to the type of errors likely to be encountered. Moreover, the Jaccard scheme allows for privacy preserving record linkage using concatenated 1-bit minwise hashing.

4.5 Record linkage summary

Much existing theory has been summarised. The general approach has been based of Fellegi-Sunter theory. It has been shown that the Fellegi-Sunter decision rule approach can be implemented efficiently and used for assessing blocking schemes. It has also been shown that blocking can be achieved on the basis of Jaccard scores, calculated from concatenated 1-bit hashes, using a similar approach to that of Bachteler et al. (2013).

The most important aspect of this work is the use of a multinomial EM approach using categorized similarity scores. It is essentially automatic, grounded in theory, and generates more meaningful posterior probabilities than methods based on applying *post hoc* weights. Thus theory can be employed in deciding thresholds. It outperforms schemes based on binary comparisons with / without *post hoc* weights, although there is some increased computational cost.

One aspect of record linkage that has not been discussed is the 1 to 1 matching of partial records. Clearly there can be at most a single correct match in File B for any given record in File A. Eliminating multiplicities using approaches such as the linear sum assignment algorithm can improve record linkage (Winkler, 1994). However, it is a *post hoc* procedure, and does not impact upon the previous discussion. It should certainly be considered as part of a comprehensive record linkage approach.

5 General summary

A novel approach to string anonymization has been presented. It is based an existing approach for identifying similar documents in large text corpora. It essentially takes a b-bit minwise hashing approach – concatenating a sequence of 1-bit hashes to generate a single hash for a set of tokens (representing a string). It is a particularly secure form of locality sensitive hashing from the point of view of frequency attacks on the tokens. Being based on XOR-ing randomly generated integers it is also robust against dictionary attacks. Similarity scores for pairs of token sets can be computed efficiently from the corresponding hashes. The Jaccard score can be expressed in terms of the hamming distance, for which there exists a particularly efficient algorithm. Hash lengths can be chosen to provide scores with a given degree of precision. Alice and Bob only need to use a shared seed (and the same PRNG / ordering of key variables).

For record matching a multinomial EM approach is used. It is easy to implement and can deal with missing values in a straightforward manner. It has been shown to provide better predictive performance than the more usual binary comparison approach, even when similarity scores are used as weights. Using the multinomial EM approach with Jaccard scores did not perform as well as when using the same approach with Jaro and Jaro-Winkler scores. However, the tests were particularly well suited to string comparators designed to cope with typographical errors. In practice, there is no reason why the same similarity score should be used for each variable. The important point is that Jaccard scores can be used in a privacy preserving manner. The use of Jaro or Jaro-Winkler requires a trusted Carol.

Blocking, and the relationships between thresholds and error rates have been discussed in some detail. The detailed implementation of an approach to relate error rates and thresholds has been presented. This is based on an algorithm contained in Fellegi and Sunter's 1969 paper. It offers a means of evaluating alternative blocking schemes. The multinomial EM approach provides a natural way of allowing blocking on the basis of similarity score thresholds. Details of an approach for identifying record pairs that exceed similarity scores (on each variable) have also been presented. Simply concatenating hash values allows blocking on record level similarity.

In summary, this report describes novel approaches for all the major aspects of privacy preserving record linkage. In particular, concatenated 1-bit hashes appear to particularly secure, and multinomial EM has been shown to be superior to the more usual binary EM with *post hoc* similarity score weighting. It also produces more meaningful posterior probabilities, allowing the use of classification thresholds based on theory.

References

Bachteler, T., Reiher, J. and Schnell, R. (2013) Similarity Filtering with Multibit Trees for Record Linkage, German Record Linkage Center, Working Paper Series NO. WP-GRLC-2013-01

Broder, Andrei Z. (1997) *On the resemblance and containment of documents*, Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, IEEE, pp.21-29

Churches, T. and Christensen, P. (2004) *Some methods for blindfolded record linkage*, BMC Medical Informatics and Decision Making

Contiero, P., Tittarelli, A., Tagliabue, G., Maghini, A., Fabiano, S., Crosignani, P. and Tessandori, R. (2005) *The EpiLink Record Linkage Software : Presentation and Results of Linkage Test on Cancer Registry Files.* Methods Inf Med, 44, pp.66-71

Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977) *Maximum Likelihood from Incomplete Data via the EM Algorithm.* Journal of the Royal Statistical Society, Series B 39 (1) pp.1-38

Durham, E.A. (2012) *A framework for accurate, efficient private record linkage.* PhD dissertation, Faculty of the Graduate School of Vanderbilt University, May 2012, Nashville, Tennessee

Elmagarmid, A.K., Ipeirotis, P.G. and Verykios, V.S. (2007) *Duplicate record detection: A survey.* Knowledge and Data Engineering, IEEE Transactions on 19 (1) pp.1-16

Fellegi, I.P. and Sunter A.B. (1969) *A theory for record linkage.* JASA Vol. 64, No. 238, pp.1183 1210

Hernández, M. and Stolfo, S. (1998) *Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem.* Data Mining and Knowledge Discovery, 2, pp.9-37

Jaro, M.A. (1976) UNIMATCH: A Record Linkage System: User's Manual, Technical Report, U.S. Bureau of the Census, Washington, D.C.

Jaro, M.A. (1989) *Advances in record linkage methodology as applied to the 1985 census of Tampa Florida.* Journal of the American Statistical Association 84 (406) pp.414-20

Kim, H. and Lee, D. (2010) HARRA: fast iterative hashed record linkage for large-scale data collections; in: Proceedings of the 13th International Conference on Extending Database Technology, Lausanne, Switzerland, New York: ACM, pp.525-536

Kirsch, A. and Mitzenmacher, M.(2006) Less hashing same performance: building a better bloom filter. In: Yossi Azar and Thomas Erlebach, editors, Algorithms-ESA 2006. Proceedings of the 14th Annual European Symposium: 11-13 September 2006; Zürich, Switzerland, pp.456-467, Berlin, Springer

Kristensen, T.G., Nielsen, J. and Pedersen, C.N.S. (2010) *A tree-based method for the rapid screening of chemical fingerprints.* Algorithms for Molecular Biology 5:9

Li, P, and König, A.C. (2011) *Theory and applications of b-bit minwise hashing*, Communications of the ACM, Vol. 54 No. 8, pp.101-109

McCallum, A., Nigam, K. and Ungar, L. (2000) *Efficient clustering of high-dimensional data sets with application to reference matching.* In Proc. of the sixth ACM SIGKDD Int. Conf. on KDD, pp.169-178

Patrascu, M. and Thorup, M. (2011) *The power of simple tabulation hashing*, Proceedings of the 43rd annual ACM symposium on Theory of computing, pp.1-10

Schnell, R., Bachteler, T. and Reiher, J. (2009) *Privacy-preserving record linkage using Bloom filters*, BMC Medical Informatics and Decision Making

Swamidass, S. J. and Baldi, P. (2007), *Mathematical correction for fingerprint similarity measures to improve chemical retrieval*, Journal of chemical information and modeling (ACS Publications) **47** (3): pp.952-964

Wegman, M.N. and Carter, L. (1981) *New classes and applications of hash functions*, Journal of Computer and System Sciences, 22(3) pp.265-279

Wegner, P. (1960) *A technique for counting ones in a binary computer.* Communications of the ACM 3 (5) pp.322

Winkler, W. E. (1990) *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.* Proceedings of the Section on Survey Research Methods (American Statistical Association) pp.354-359

Winkler, W. E. (1992) *Comparative Analysis of Record Linkage Decision Rules.* Proceedings of the Section on Survey Research Methods, American Statistical Association, pp.829-834

Winkler (1994) *Advanced Methods for Record Linkage.* Proceedings of the Section on Survey Research Methods, American Statistical Association pp.467-472

Winkler, W. E. (1995). *Matching and Record Linkage.* In: B. G. Cox et. al. (ed.) Business Survey Methods. New York, J. Wiley, pp.355-384

Yancey W.E. (2002) *Improving EM Algorithm Estimates for Record Linkage Parameters.* http://www.amstat.org/sections/SRMS/proceedings/y2002/Files/JSM2002-000581.pdf

Appendices

A Revised decision rule algorithm

For larger numbers of matching variables, and when the binary comparison assumption is dropped, the computational costs of the decision rule algorithm might be excessive. Ideally we would like a means of iterating over the ordered γ without having to construct the complete ordering.

Note that the values $m(\gamma_n)/u(\gamma_n)$ and $m(\gamma_{n'})/u(\gamma_{n'})$ define thresholds that allow us to restate the decision rule without reference to indices. Thus we do not need the complete ordered set of comparison vectors for classification purposes. We simply need to be able to iterate over the ordered set as far as n , and over the ordered set in reverse order as far as n' .

The following depends on the naïve Bayes assumption:

Firstly we can simplify this somewhat. When iterating over the γ in decreasing order we are summing over the $u_i(\gamma)$. For any comparison vector containing a variable level with $u = 0$ then $u_i(\gamma) = 0$. So we need only generate comparison vectors that contain no such terms. When iterating over the γ in increasing order we are summing over the $m_i(\gamma)$. We need not generate comparison vectors that contain any terms with $m = 0$. Thus we achieve the exclusion of γ for which $m(\gamma) = u(\gamma) = 0$. The question is whether we can exclude all variable levels with either $u = 0$ or $m = 0$.

For $\mu > 0$ we must include at least one γ with $u(\gamma) > 0$. This will have to have $m(\gamma) > 0$ unless we have a degenerate situation where the comparison vector with largest likelihood ratio has likelihood ratio 0.

For $\lambda > 0$ we must include at least one γ with $m(\gamma) > 0$. This will have to have $u(\gamma) > 0$ unless we have a degenerate situation where the comparison vector with smallest likelihood ratio has infinitely large likelihood ratio.

So iteration over the ordered comparison vectors in either decreasing or increasing orders will involve visiting at least one vector with non-zero, finite likelihood. For admissible μ and λ $n < n' - 1$ so iteration must also stop at such a value.

Consider iterating over the γ in decreasing order of $m(\gamma)/u(\gamma)$. For each variable create a list containing 3-tuples (m/u, m, u) of values associated with the variable levels, excluding any levels for which $m = 0$ or $u = 0$. Sort each list in decreasing order (of m/u) and append to another list.

$$\begin{aligned}&[[p_{00},\ p_{01},\ p_{02}, p_{03}, p_{04}],\\&\ [p_{10},\ p_{11},\ p_{12}],\\&\ [p_{20},\ p_{21},\ p_{22},\ p_{23}],\\&\ [p_{30},\ p_{31}]]\end{aligned}$$

p_{xy} above is the tuple, (m/u, m, u), for the yth level of the xth variable. Note: under the binary comparison assumption each sub-list would have length 2. Under the multinomial scheme the sub-lists can contain different numbers of categories.

The comparison space is the Cartesian product of the sub-lists. Each comparison vector can be signified by a tuple of indices into the lists. Thus (0,0,0,0) denotes the first elements in each list above (using 0-based indexing). The sub-lists contain sequences of monotone non-increasing values. So the likelihood ratio $m(\gamma)/u(\gamma)$ associated with an index tuple γ is greater than, or equal to, the likelihood ratio associated with an index tuple that can be generated by incrementing (one or more of) the indices in γ. Thus the likelihood ratio associated with (0,0,0,0) is greater than, or equal to, the likelihood ratio associated with e.g. (0,1,0,0). We can construct a graph of the (reduced) comparison space with index tuples as nodes. Edges signify the transitive relation "at least as great as". The single source node is the index tuple containing zeros. The children of a node are generated by incrementing each of the tuple values subject to the constraints implied by the sub-list lengths. Figure 9 shows the graph generated from two key variables with categorizations of length 2 and 3 respectively.

The graph encodes many relationships between the likelihood ratios, but not for instance, the relationship between nodes (0,1) and (1,0). So we have a partial ordering implied by the graph which e.g. dictates that there is an ordering of the nodes such that nodes (0,1) and (1,0) can be ordered earlier than (1,1). The relative ordering of (0,1) and (1,0) requires inspection of their likelihood ratios.

Relationships that cannot be resolved by inspection of the graph can be resolved by pushing nodes containing the relevant data onto a priority queue. We can limit the size of the priority queue by only placing nodes in the

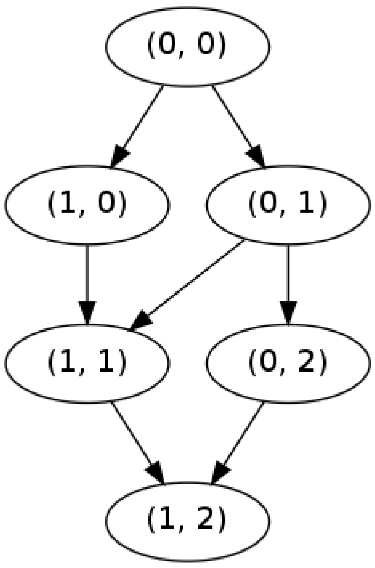

Figure 9: Transitive graph for two key variables of length 2 and 3

queue when they become candidates for the "next largest value". The transitive property dictates that a node does not become a candidate (for the next largest likelihood ratio) until all its parents have been generated.

We do not want to produce the complete graph, because we might only need a small number of nodes in order to produce the relevant threshold. Thus we would like to generate nodes lazily (on the fly). Generating the children of a node is straightforward, although we would like to generate children only once, negating the need to keep track of which children have already been generated and queued / processed. One method for achieving this is to ensure that a node is only generated by its parent that is lowest in a lexicographic ordering of its parents. Thus (0,1) would generate (1,1), and (1,0) would generate no children. In effect, we only exploit the transitive relationships contained in a spanning tree of the graph, allowing the priority queue to handle the rest. A better alternative would be to only generate a child when the last of its parents was popped from the priority queue - but there is no obvious efficient way of achieving this. The following Python code demonstrates that generating children according to the smallest lexicographic ordered parent scheme is relatively simple.

```
def children(node, shape):
    # node is a list e.g.  [0,2,1]
    # shape contains lengths of lists e.g.  [4,5,5]
    # lex smallest parent
    for i, val in enumerate(node):
        if val + 1 < shape[i]:
            new_node = list(node)
            new_node[i] += 1
            yield new_node
        if val:
            break
```

The initial node is pushed onto the priority queue. It consists of a 3-tuple ($m(\gamma)/u(\gamma)$, $m(\gamma)$, $u(\gamma)$) and its index (0, . . . , 0). Nodes are then popped from the queue and processed. Processing consists of maintaining the cumulative sum of u_i probabilities, generating children and the tuples of data associated with them, and pushing them onto the priority queue.

For finding n' we need to iterate over the comparison vectors in increasing order of $m(\gamma)/u(\gamma)$. This is achieved using almost exactly the same approach. The ordering of the sub-lists is reversed, the priority of the priority queue is reversed, and the m_i are summed.

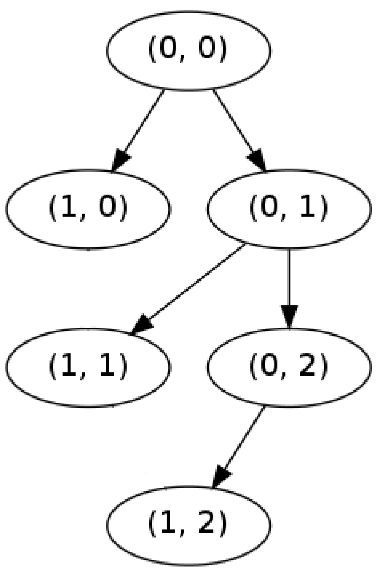

Figure 10: Lexicographic ordered parent scheme spanning tree for the graph in the previous figure

Using the above scheme it is not guaranteed that the nodes generated in decreasing order will be the reverse of the nodes generated in increasing order. But for admissible μ and λ the two sequences will be non-overlapping unless the comparison vectors with indices n and n' have equal likelihood ratios. But this is an issue that we already have to deal with in calculating P_μ and P_λ .

From earlier,

$$u_n P_\mu = \mu - \sum_{i=1}^{n-1} u_i.$$

In this case we might have distinct comparison vectors with equal likelihood ratios that are treated differently. Those ordered earlier would result in allocation to A_1 whilst the comparison vector with index i would result in an allocation to A_1 with probability P_μ . If we choose not to distinguish between comparison vectors with equal likelihood ratios, then we can treat them as a single comparison vector with $m(\gamma)$ and $u(\gamma)$ equal to the sum of the corresponding terms for the individual comparison vectors. As far as the implementation is concerned we simply need to maintain sums for each distinct likelihood ratio (floating point issues will not be a problem as the ratios are generated identically for decreasing and increasing orderings). In this way the above approach can be guaranteed to generate decreasing orderings that are the reverse of the corresponding increasing orderings.

Note that when the comparison space is very large we can sample (with replacement) from the comparison space and construct a list of comparison vectors. This can be sorted and iterated over to generate approximate solutions to the above problems.

B Blocking implementation

An alternative to blocking on the basis of a similarity score threshold for a comparison vector, is to specify a distinct threshold for each of the blocking variables. Only record pairs that exceed each threshold are considered as possible links. When using concatenated 1-bit hashes these thresholds can be converted to thresholds on the hamming distance / number of colliding bits – as shown earlier.

The basic data structure is a trie. A trie[5] is a tree structure that contains a set of sequences. It has a value associated with each edge, and each node v is associated with the sequence of values on the unique path from the source node to v (although this sequence is not necessarily stored in the node). A trie is generally constructed iteratively, adding a single sequence at a time. The sequence is iterated over, and the corresponding path from the root of the trie followed. New nodes and edges are added as required.

The first step is to construct mappings of the distinct comparison vectors in A to the relevant record IDs in A, and the distinct comparison vectors in B to the relevant record IDs in B. Thus when similar comparison vectors are found, the Cartesian product of the relevant record ID lists are possible matches. A trie is constructed containing the keys (distinct comparison vectors) in the mapping for A, and then searched for similar comparison vectors for each key in the mapping for B.

Figure 11 shows a trie for blocking variables gender, age and first name for 6 distinct comparison vectors. Strings are shown, rather than hash values, for the purpose of clarity. Note that here name has been reduced to the first two letters. This is purely for blocking purposes, and would require that Alice and Bob generate an additional variable to be used solely for blocking. Each sink node w contains a reference to the comparison vector represented by the edges on the path from the source node to w.

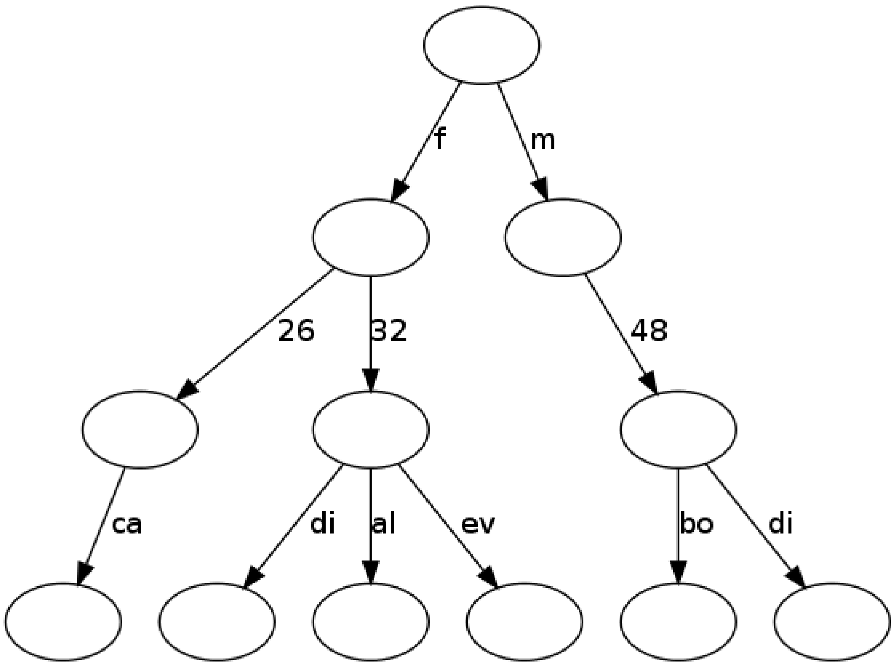

Figure 11: Trie data structure containing sequences

Checking for set membership consists of a search from the root. For instance, the record ['f', 28, 'do'] would result in a search from the root, finding 'f' but not finding an outgoing edge from the child of 'f' with value 28 – so the record is not in the trie. The trie is implemented so that membership of the outgoing edges from a node can be tested (and the relevant edge returned, if present) in constant time.

Blocking on exact matches is simple to implement. If a comparison vector from File B is found in the trie, then we have record pairs (a, b) that match exactly. But we would not want to exclude record pairs that match on a subset of variables if the remaining variables contain missing values. The pairs match on all the values that are available for comparison. In order to accommodate this we reserve a value (the empty string) to denote a missing value. This is assumed to match any value. Thus the record ['f', '32', ''] would be considered to be a possible match against the records for 'al(ice)', 'di(ana)' or 'ev(e)' – but not 'di(ck)' who is male and

[5]http://en.wikipedia.org/wiki/Trie

48. In fact, this is the primary reason that a trie is used – alternatives, such as generating keys by hashing comparison vectors, do not allow the handling of missing values. So we are actually performing partial matching by potentially searching along multiple paths for a given target vector.

Extending this to handle similarity scores is straightforward. Again we might search along multiple paths, but now the paths are determined by similarity rather than exact matching (or assumed matching in the presence of missing values). At each node we only search along branches such that the similarity score on the relevant edge exceeds a predefined threshold. In a non privacy preserving context we could simply use the string values, but in a privacy preserving context concatenated 1-bit hashes are used.

The multibit tree structure used by Bachteler et al. (2013), and due to Kristensen et al. (2010), is basically a trie constructed using sequences of bits. The use of a multibit tree allows the search to be pruned, by not searching branches that can only lead to bitstrings that are sufficiently dissimilar to the target bitstring. This requires the calculation of bounds on the Jaccard score at each bit position visited. Once the upper bound is below the relevant threshold the search along a particular branch can be halted, in the same way that we could stop searching Figure 11 after the first level if looking for an exact match for a target record ['f', 37, 'bo'].

We use a similar approach, in that each set of outgoing edges from a node w is actually a multibit tree, with sink nodes equal to the children of w. A threshold on the similarity score is converted to a threshold on the hamming distance,

$$h_{max} = \left\lfloor \frac{m(1 - J_{min})}{2} \right\rfloor$$

and hence to a threshold on the number of colliding bits,

$$N_{min} = m - h_{max}.$$

Generation of lower and upper bounds on the number of colliding bits is trivial. Calculations are much simpler than those required for generating bounds on the (estimated) Jaccard score given two Bloom filters (see Kristensen et al., 2010, for details). However, it should be noted that concatenated 1-bit hashes tend to be a little longer than Bloom filters for a given degree of precision in Jaccard score estimates, so their multibit trees tend to be deeper.

It is also trivial to place confidence intervals on the total number of collisions, or indeed on the underlying Jaccard score (below), given the number of collisions over any sample or sub-sequence of n bits,

$$\left(2\pi_n - 1 - z_{\frac{\alpha}{2}} 2\sqrt{\frac{\pi_n(1-\pi_n)}{n}}, 2\pi_n - 1 + z_{\frac{\alpha}{2}} 2\sqrt{\frac{\pi_n(1-\pi_n)}{n}}\right)$$

where π_n is the fraction of collisions over the n bits.

There is a small sequential testing issue here, but classifying on the basis of confidence intervals can significantly reduce search costs. It should be noted that although the approach based on bounds is exact, it only relates to the Jaccard score estimate based on the full bitstrings. The confidence interval above relates to the underlying Jaccard score, rather than an estimate.

Given the trie data structure with multibit trees / binary tries as outgoing edge sets, we have a number of options for blocking. We can specify a distinct threshold for each variable. Thus if the values for gender were sufficiently dissimilar in Figure 11, then we could stop searching very early. We could require that these thresholds had to be exceeded for a given number of variables, in which case search would proceed until a sufficient number of thresholds had not been satisfied. We can concatenate (and perhaps compress) the individual hashes for variables in order to generate hashes for comparison vectors, in which case the approach is more similar to that of Bachteler et al. (2013). The data structure could be used to reduce the computational costs of canopy clustering, although the potential benefits of this are still to be assessed.