Proceedings of Statistics Canada Symposium 2010
Social Statistics: The Interplay among Censuses, Surveys and Administrative Data

An Empirical comparison of approaches to approximate string matching in private record linkage

Tobias Bachteler, Rainer Schnell, and Jörg Reiher[1]

Abstract

Due to the frequency of spelling and typographical errors in practical applications, record linkage algorithms have to use string similarity functions. In many legal contexts, identifiers such as names have to be encrypted before a record linkage can be attempted. Therefore, algorithms for computing string similarity functions with encrypted identifiers are essential for approximating string matching in private record linkage. This study reports an empirical evaluation of three promising approaches to compute similarities between identifiers in a privacy preserving manner.

Key Words: Approximate String Matching; Privacy-preserving Record Linkage; Private Record Linkage

1. Introduction

Combining multiple databases with additional information on the same person is increasingly occurring throughout research. Whenever feasible, the databases are merged using a unique identification number. Otherwise, probabilistic record linkage is most frequently applied for the identification of matching record pairs. However, in many applications the identifiers have to be encrypted due to privacy concerns, which is problematic because linking encrypted identifiers can result in serious complications. The problem of finding records that represent the same individual in separate databases without revealing the identity of the individuals is called the "privacy-preserving record linkage", "blind data linkage", or "private record linkage" problem.

Due to the frequency of spelling and typographical errors, in practical applications record linkage algorithms have to use string similarity functions. In addition, since records with variations of identifiers may have different characteristics than records with exact matching identifiers, restricting the linkage in this manner is not an option. Therefore, algorithms for computing string similarity functions with encrypted identifiers are essential for approximate string matching in private record linkage. This study reports an empirical evaluation of three promising approaches to compute similarities of identifiers in a privacy preserving manner. The performances of these protocols were compared with each other and those of an edit-distance applied to unencrypted identifiers.

1.1 Problem Statement

The problem of how to accomplish approximate string matching in private record linkage can be compactly stated as follows: Consider two database owners *A* and *B* holding lists of strings (e. g. names) $S_a = \{a_1, \ldots, a_n\}$ and $S_b = \{b_1, \ldots, b_m\}$. Compute string similarities of all pairs $(a_i, b_j) \in S_a \times S_b$ such that all the names remain private to the database holders.

1.2 Candidate Solutions

Since the early 1990s several intriguing approaches to approximate string matching in private record linkage were proposed. Most of these (e.g. Bouzelat et al., 1996, Atallah et al., 2003, Ravikumar et al., 2004, and Churches and Christen, 2004) face problems, for instance very high computing demands or high rates of false positive links. These

[1] Tobias Bachteler (tobias.bachteler@uni-due.de); Rainer Schnell (rainer.schnell@uni-due.de); Jörg Reiher (joerg.reiher@uni-due.de), University of Duisburg-Essen, Lotharstr, 65, D-47057 Duisburg, Germany

problems make them unfeasible in realistic record linkage settings. There are however three approaches to compute similarities of identifiers in a privacy preserving manner, which *prima facie* are promising candidate solutions to the problem, namely the protocols proposed by Pang and Hansen (2006), Scannapieco et al. (2007), and Schnell et al. (2009).

Pang and Hansen (2006) suggested a protocol based on a table of reference strings, *R*, common to *A* and *B*. The elements of *R* should reflect the domain of the strings to be matched. For a given identifier, both database holders compute the edit-distances, *d*, between each identifier string and all reference strings in the set. If *d* is equal or less than a threshold δ, the respective reference string is encrypted using a key previously agreed on by *A* and *B*. For each identifier string, the resulting set of encrypted reference strings along with their distances, *d*, and an ID number form a tuple. Both database holders send their tuples to a third party *C*. For every pair of ID numbers where the encrypted reference strings agree, *C* sums the distances, *d*, and finds the minimum of this sum, which serves as an approximation to the distance between the plain text identifiers.

In the protocol of Scannapieco et al. (2007), two data holders, holding lists of names, build an embedding space from random strings and embed their respective strings therein using the SparseMap method (Hristescu and Farach-Colton, 1999). Then, each data holder sends the embedded strings to a third party who determines their similarity. To create the embedding space, data holder *A* generates *n* random strings of length *l* and builds *k* reference sets from them. These *k* reference sets are used to embed the names in a *k*-dimensional space. The coordinates for a given name are approximations of the distances between the name to the closest random string in each of the *k* reference sets in terms of the edit-distance. As a result, for each name *A* receives a *k*-dimensional vector. After receiving the *k* reference sets from *A*, *B* embeds his names in the same way. Finally, both data holders send their vectors to a third party, *C*, who compares them using the Euclidean distance between them.

Schnell et al. (2009) proposed to take advantage of the cryptographic properties of Bloom filters (Bloom, 1970) to determine string similarities in a privacy preserving manner. A Bloom filter is a bit array with all bits initially set to 0. To store a string in a Bloom filter, each of its constituent *n*-grams are hashed using *k* HMACs (Bellare et al., 1996) and all bit positions that correspond to a hash value are set to 1. The Bloom filters of two similar strings will share a good proportion of bit positions set to 1. First, data holders *A* and *B* store each name in a separate Bloom filter and transfer them to a third party *C*. Then, *C* is able to approximate the *n*-gram similarity of the original names by computing the Dice coefficient $D = 2h/(m+n)$ where *h* is the number of bit positions conjointly set to 1 in both filters, *m* is the number of bit positions set to 1 in the first and *b* the number of bit positions set to 1 in the second Bloom filter.

2. Methods

2.1 Test Data

For empirical testing we used experimentally generated human data. 1,500 surnames were randomly selected from a town register. Then, 15 audio tapes with 100 different names each were recorded with a female voice. Each tape was played to one of 15 groups of students with an average of 19 students per group. Each student wrote the name heard, 13 groups by hand, two groups typed the names. Finally, all student notes were typed by a single typist resulting in 1,286 unique original names (file A) and 27,296 potentially erroneous names (file B).

To compare the three candidate solutions to the problem of approximate string matching in private record linkage we determined the similarity of each pair of names $(a,b) \in A \times B$ using each method in turn and determined their performance by examining precision-recall plots. As an external standard of comparison we determined the Damerau-Levenshtein distance from the plain text names.

2.2 Precision-Recall Plots

As is the norm in the information retrieval literature, the criteria of *recall* and *precision* were used to determine the linking effectiveness of the method. For a given level of similarity φ, a pair of records is considered as a match if the pair is actually a true pair, all other pairs are called non-matches. Based on the common classification for true positive (*TP*), false positive (*FP*), false negative (*FN*) and true negative (*TN*) pairs, the comparison criteria are defined as

$$recall = \frac{\sum TP}{\sum TP + \sum FN} \quad (1)$$

$$precision = \frac{\sum TP}{\sum TP + \sum FP} \quad (2)$$

Plotting precision and recall for different similarity values φ as a curve in a precision-recall-plot shows the performance of a string comparison method. A procedure with a better performance will have a curve in the upper right of the plot.

3. Results

3.1 Protocol 1 (Pang and Hansen)

To evaluate the protocol of Pang and Hansen (2006) we varied two parameters: the threshold δ and the size of the reference table *R*. To obtain suitable reference strings, we sampled surnames from a German phone book.

Figure 3.1-1
Protocol 1: Precision-recall curves using thresholds 2 and 3 with sample sizes 5,000 and 10,000

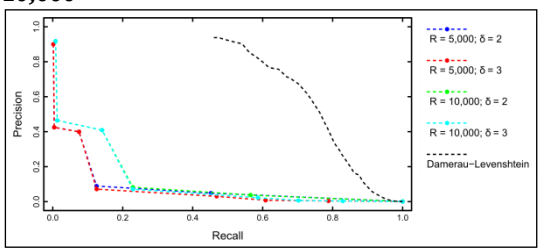


Figure 3.1-2
Protocol 1: Precision-recall curves using threshold 2 and various sample sizes

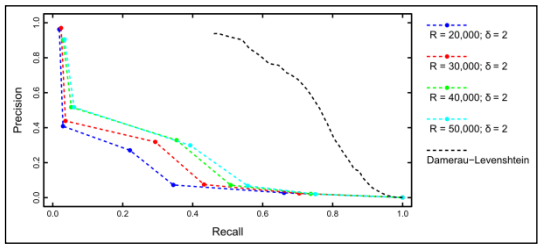


Figure 3.1-1 displays precision and recall for reference tables of size 5,000 and 10,000 with $\delta = 2$ and $\delta = 3$ respectively. Due to protocol design, there are just a few precision-recall points obtainable, which are marked by dots and connected by dashed lines. All curves indicate that the respective parameter settings are performing fairly poor. Starting at a recall of virtually null, precision drops abruptly and reaches a low level of precision before recall starts to rise significantly. That is, the protocol is not able to differentiate between matching and not matching pairs of names at all. This finding is reinforced by the huge gap between the precision-recall points and the reference line of the Damerau-Levenshtein distance.

As figure 3.1-2 shows, increasing the size of the reference table improves the situation to some extent. However, as the sample size gets larger, the additional gain diminishes significantly. Choosing a size beyond 50,000 does not improve the situation further. The use of $\delta \geq 3$ along with the increased sample sizes is impractical since this leads to very long run times. In all, the best performance is obtained with the parameter combination $\delta = 2$ and $R = 50,000$.

3.2 Protocol 2 (Scannapieco et al.)

In the protocol of Scannapieco et al. (2007), two parameters are of interest, namely the length of the random strings *l* and the number of dimensions *k*. As a starting point we used recommendations given by Scannapieco et al. They suggested a dimensionality of about 20 and string length corresponding to length of input names, which on average is about 7 in our test data base. Because there is randomness built in the protocol via the random strings in the reference sets, we replicated each individual test ten times. The precision-recall curves show the of average recall at 10 fixed precision values. Minimum and maximum recall are marked by caps at each precision value.

Figure 3.2-1
Protocol 2: Precision-recall curves using 25 dimensions and various string lengths

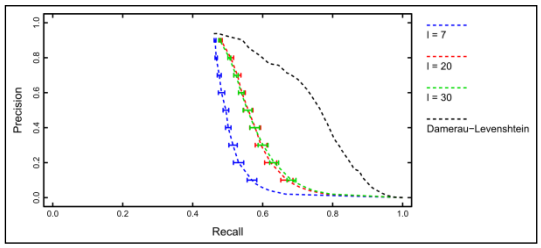


Figure 3.2-2
Protocol 2: Precision-recall curves using string length 20 and various numbers of dimensions

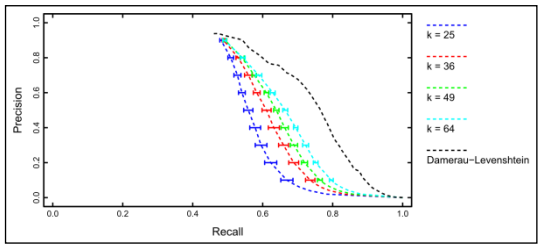


Figure 3.2-1 shows precision and recall depending on different string lengths (number of dimensions fixed at $k = 25$). All curves have some distance from the reference line indicating a rather poor performance of the parameter settings. However, contrary to the expectations, the string lengths 10 and 20 do definitely better than a string length of 7.

In a second experiment, we varied the number of dimensions, *k*, at a fixed string length of 20. As shown in Figure 3.2-2, using more dimensions generally leads to improved performances of the protocol. However, as can be seen from the distances between the individual curves, the additional gain attained is diminishing with increasing numbers of dimensions. As the best performing parameter combination we identified a string length of $l = 20$ along with a dimensionality of $k = 64$.

3.3 Protocol 3 (Schnell et al.)

For an evaluation of the protocol of Schnell et al. (2009), the parameter of main interest is the number of hash functions, *k*. We compared performances of different *n*-gram variants. In all experiments, the lengths of the bit arrays were fixed at 1,000 bits. In Figure 3.3-1, the precision-recall curves resulting from 2-grams and different numbers of hash functions are displayed. The best performance is attained with 10 hash functions. Although increasing this number generally results in inferior performances, even the curve resulting from 70 hash functions indicates an acceptable result.

Figure 3.3-1
Protocol 3: Precision-recall curves using 2-grams and various numbers of hash functions

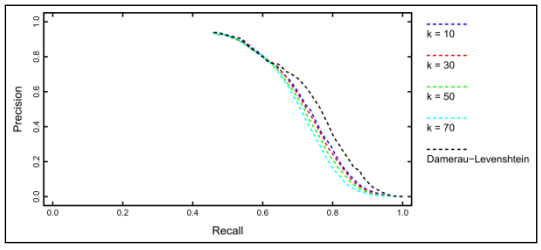


Figure 3.3-2
Protocol 3: Precision-recall curves using 50 hash functions and various variants of *n*-grams

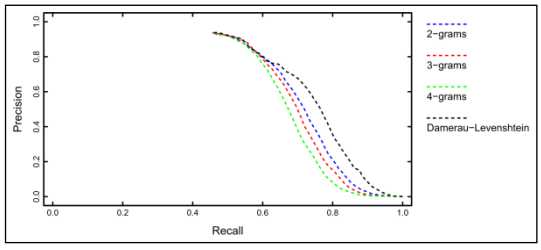
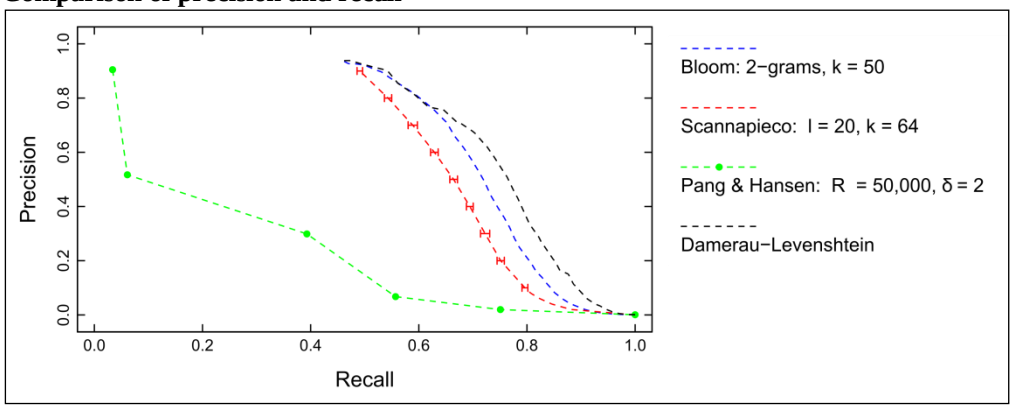


Figure 3.3-2 shows the precision-recall curves of 2-, 3-, and 4-grams with *k* fixed at 50 hash functions. The best performing variant are 2-grams, followed by 3-grams. In all, the best of the evaluated parameter combinations in terms of precision and recall is 2-grams with $k = 10$. However, since the use of more hash functions provides more security (Schnell et al., 2009), we selected 2-grams with $k = 50$ for the inter-protocol comparison.

3.4 Comparison

For comparison we used the best performing variants of Pang and Hansen and Scannapieco et al. along with the 2-grams, $k = 50$ variant of Schnell et al. The primary comparison criterion is the effectiveness of the protocols in terms of precision and recall. Additionally, we compared the run times.

Figure 3.4-1
Comparison of precision and recall



As shown in Figure 3.4-1, the Bloom filter method outperforms the protocol of Scannapieco et al. and both outperform the protocol of Pang and Hansen substantially. The small gap between the curves of the Bloom filter method and Damerau-Levenshtein indicates that the Bloom filter method performs quite well compared with a similarity function applied to unencrypted strings.

The run times of the various methods are given by table 3.4-1. The protocol of Pang and Hansen requires substantial run times both to encrypt and compare the names. The total run time of nearly 16 hours implies that the protocol will be virtually impractical for most matching tasks. In contrast, the run times of the Scannapieco method and the Bloom filter method are quite reasonable. Due to very efficient encryption, the Bloom filter method did not require substantially more time to compare the names than Damerau-Levenshtein.

Table 3.4-1
Comparison of run times (minutes)

Method	Encryption	Matching	Total
Damerau-Levenshtein	-	47:15	47:15
Bloom	0:04	52:56	53:00
Scannapieco (means)	19:25	65:25	84:50
Pang and Hansen	470:14	494:13	964:27

4. Conclusions

In this article, we report the results of an empirical evaluation of three promising approaches to compute similarities between identifiers in a privacy preserving manner. For empirical testing we used experimentally generated human data. We identified the best parameter settings of the three methods. The performance of these methods was compared. The protocol proposed by Pang and Hansen shows poor precision and recall and requires long run times. The protocol of Scannapieco et al. yielded good linkage results and a moderate run time. The protocol proposed by Schnell et al. showed the best linkage results and lowest run time.

References

Atallah, M. J., F. Kerschbaum, and W. Du (2003), "Secure and Private Sequence Comparisons", *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, pp. 39-44.

Bellare, M., R. Canetti, and H. Krawczyk (1996), "Keying Hash Functions for Message Authentication", *Proceedings of the 16th Annual International Cryptology Conference*, pp. 1-15.

Bloom, B. H. (1970), "Space/time Trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM*, 13, pp. 422-426.

Bouzelat, H., C. Quantin, and L. Dusserre (1996), "Extraction and Anonymity Protocol of Medical File", *Proceedings of the 1996 AMIA Annual Fall Symposium*, pp. 323-327.

Churches, T. and P. Christen (2004), "Some Methods for Blindfolded Record Linkage", *BMC Medical Informatics and Decision Making*, 4.

Hristescu, G., and M. Farach-Colton (1999), "Cluster-preserving Embedding of Proteins", *DIMACS Technical Report* 99–50, DIMACS Center for Discrete Mathematics & Theoretical Computer Science.

Pang, C., and D. Hansen (2006), "Improved Record Linkage for Encrypted Identifying Data", *Proceedings of the 14th Annual Health Informatics Conference*, pp. 164-168.

Ravikumar, P., W. W. Cohen, and S. E. Fienberg (2004), "A Secure Protocol for Computing String Distance Metrics", paper presented at the Workshop on Privacy and Security Aspects of Data Mining, Brighton, England.

Scannapieco, M., I. Figotin, E. Bertino, and A. K. Elmagarmid (1996), "Privacy Preserving Schema and Data Matching", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 653-664.

Schnell, R., T. Bachteler, and J. Reiher (2009), "Privacy-preserving Record Linkage using Bloom Filters", *BMC Medical Informatics and Decision Making*, 9.