Automatic Record Linkage using Seeded Nearest Neighbour and Support Vector Machine Classification

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

ABSTRACT

The task of linking databases is an important step in an increasing number of data mining projects, because linked data can contain information that is not available otherwise, or that would require time-consuming and expensive collection of specific data. The aim of linking is to match and aggregate all records that refer to the same entity. One of the major challenges when linking large databases is the efficient and accurate classification of record pairs into matches and non-matches. While traditionally classification was based on manually-set thresholds or on statistical procedures, many of the more recently developed classification methods are based on supervised learning techniques. They therefore require training data, which is often not available in real world situations or has to be prepared manually, an expensive, cumbersome and time-consuming process.

A novel two-step approach to automatic record pair classification has previously been presented by the author. In its first step, training examples of high quality are automatically selected from the compared record pairs, and used in the second step to train a support vector machine (SVM) classifier. Initial experiments showed the feasibility of this approach, achieving results that outperformed k-means clustering. Two variations of this approach are presented in this paper. The first is based on a nearest-neighbour classifier, while the second improves a SVM classifier by iteratively adding more examples into the training sets. Experimental results show that this two-step approach can achieve better classification results than other unsupervised approaches.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; H.2.8 [**Database Management**]: Database applications—*Data mining*

General Terms

Algorithms, Experimentation, Performance

Submitted to KDD 2008 Las Vegas

Keywords

Data matching, data linkage, deduplication, entity resolution, nearest neighbour, support vector machine

1. INTRODUCTION

As increasingly large amounts of data are being collected by many organisations, techniques that enable efficient mining of massive databases have in recent years attracted interest from both academia and industry. Sharing of large databases between organisations is also of growing importance in many data mining projects, as data from various sources often has to be linked and aggregated in order to improve data quality, or to enrich existing data with additional information [9, 24, 25]. Similarly, detecting duplicate records that relate to the same entity within one database is commonly required in the data preparation step of many data mining projects [14]. The aim of such linkages and deduplications is to match all records that relate to the same entity. These entities can be, for example, patients, customers, businesses, product descriptions, or publications.

Traditionally, record linkage has been employed in the health sector and within statistical agencies [24]. Today, many public and private sector organisations use deduplication and linkage techniques to improve the quality of their databases. Government agencies use record linkage to, for example, identify people who register for assistance multiple times, or who collect unemployment benefits despite being employed. National security, and crime and fraud detection, are other areas where record linkage is increasingly being employed. Security agencies often require fast access to files of a particular individual in order to solve crimes or to prevent terror through early intervention [18].

If all databases to be linked contain common entity identifiers (or keys), then the problem of linking at the entity level can be solved by a standard database join. In most situations, however, no such entity identifiers are available, and therefore more sophisticated linkage techniques have to be applied. Broadly, these techniques can be classified into deterministic, probabilistic, and modern approaches [9].

Figure 1 outlines the general record linkage process. An important initial step for successful linkage is data cleaning and standardisation, as in most real-world databases noisy, incomplete and incorrect information is common [10]. A lack of high quality data can be one of the biggest obstacles to successful record linkage. The main tasks of data cleaning and standardisation are the conversion of the raw input data into well defined, consistent forms, and the resolution of inconsistencies in the way information is represented [10].

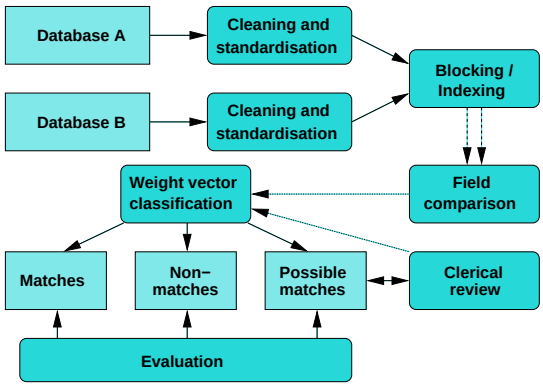

Figure 1: The general record linkage process. The output of the blocking step are candidate record pairs, while the field (attribute) comparison step generates weight vectors with matching weights.

R1:	Christine	Smith	42	Main	Street
R2:	Christina	Smith	42	Main	St
R3:	Bob	O'Brian	11	Smith	Rd
R4:	Robert	Bryce	12	Smythe	Road

WV(R1,R2): [0.9, 1.0, 1.0, 1.0, 0.9]
WV(R1,R3): [0.0, 0.0, 0.0, 0.0, 0.0]
WV(R1,R4): [0.0, 0.0, 0.5, 0.0, 0.0]
WV(R2,R3): [0.0, 0.0, 0.0, 0.0, 0.0]
WV(R2,R4): [0.0, 0.0, 0.5, 0.0, 0.0]
WV(R3,R4): [0.7, 0.3, 0.5, 0.7, 0.9]

Figure 2: Four example records (made of given name and surname; and street number, name and type attributes) and the corresponding weight vectors (WV) resulting from the comparisons of these records.

When linking two databases, **A** and **B**, potentially each record in **A** should be compared with all records in **B**. Therefore, the total number of potential record pair comparisons equals $|\mathbf{A}| \times |\mathbf{B}|$, with $|\cdot|$ denoting the number of records in a database. Similarly, when deduplicating a database, **A**, the total number of potential record pair comparisons is $|\mathbf{A}| \times (|\mathbf{A}| - 1)/2$, as each record potentially has to be compared to all others. However, as the performance bottleneck in a record linkage or deduplication system is normally the expensive detailed comparison of field (or attribute) values between records [1, 9], it is impossible to compare all pairs when the databases are large. Additionally, assuming there are no duplicate records in the databases to be linked (i.e. one record in **A** can only be a true match to one record in **B**, and vice versa), then the maximum number of true matches corresponds to $min(|\mathbf{A}|, |\mathbf{B}|)$. Thus, when linking larger databases the computational efforts potentially increase quadratically while the maximum number of true matches only increases linearly. This also holds for deduplication, where the number of true duplicate records is always less than the number of records in a database.

To reduce the potentially very large number of comparisons to be conducted between records, some form of indexing or filtering technique, collectively known as *blocking* [1], is employed by most record linkage systems. A single record attribute, or a combination of attributes, often called the *blocking key*, is used to split the databases into blocks. All records that have the same value in the blocking key will be inserted into the same block, and candidate record pairs are only generated from records within the same block. Even though blocking will remove many of the record pairs that are obvious non-matches, some true matches will also likely be removed in the blocking process because of errors or typographical variations in attribute values [9].

The two records in a candidate pair are compared using similarity functions applied to selected record attributes (fields). These functions can be as simple as an exact string or a numerical comparison, can take typographical variations into account [25], can be specialised for example for date or time values, or they can be as complex as a distance comparison based on look-up tables of geographic locations (longitudes and latitudes). There are also various approaches to learn such similarity functions from training data [3, 11]. Each similarity function returns a numerical *matching weight* that is usually normalised, such that 1 corresponds to exact similarity and 0 to total dissimilarity, with attribute values that are somewhat similar having a matching weight somewhere in between 0 and 1.

As illustrated in Figure 2, for each compared record pair a *weight vector* is formed that contains the matching weights calculated for that pair. Using these weight vectors, candidate pairs are then classified into *matches*, *non-matches*, and *possible matches*, depending upon the decision model used [9, 15, 17]. Pairs of records that are not compared due to the blocking process are implicitly assumed to be non-matches. Assuming there are no duplicate records in the databases to be linked, then the majority of candidate pairs are likely non-matches, as the maximum possible number of true matches corresponds to the number of records in the smaller of the databases that are linked. Classifying record pairs is therefore often a very imbalanced problem [9].

Two records that have equal or very similar values in all their attributes will likely refer to the same entity, as it is unlikely that two entities have very similar or even the same values in all their record attributes. All matching weights calculated when comparing such a pair of records will be 1, or close to 1. On the other hand, weight vectors that contain matching weights of only 0, or values close to 0, were with high likelihood calculated when two records that refer to two different entities were compared, as it is unlikely that two records that refer to the same entity have totally different values in all their attributes. For example, when a person moves house, most their address details will change, while their name and date-of-birth will stay the same, so there would be several record attributes that keep their values.

Based on these observations, it is often easy to accurately classify record pairs as matches when their weight vectors contain only matching weights close to or equal to 1, and as non-matches when their weights are all close to or equal to 0. On the other hand, it is more difficult to correctly classify record pairs that have some similar and some dissimilar attribute values. In Figure 2, for example, records *R1* and *R2* are very similar to each other, with only two small differences in their given name and street type attributes, and thus very likely refer to the same person. On the other hand, records *R3* and *R4* are more different from each other, and it is not obvious if they refer to the same person or not.

It follows that it is possible to automatically select training examples (weight vectors) from the set of all weight vectors that with high likelihood correspond to true matches or

true non-matches, and to then train a supervised binary classifier using these training examples as 'seeds'. For example, of the weight vectors shown in Figure 2, *WV(R1,R2)* can be selected as a match training example, while *WV(R1,R3)* and *WV(R2,R3)*, possibly even *WV(R1,R4)* and *WV(R2,R4)*, can be used as non-match training examples.

This two-step approach to automated record pair classification has first been proposed by the author in [6], with initial experiments indicating its feasibility. The contribution of this paper is the evaluation of two improved classification methods to be used in the second step of the approach. The first method is based on nearest-neighbour classification, and the second improves SVM classification by iteratively adding more weight vectors into the training sets.

The remainder of this paper is structured as follows. An overview of related work is given next. Then, the proposed two-step approach to record pair classification is presented in detail in Section 3, with the new classification methods discussed in Section 3.2. These two methods are then evaluated experimentally in Section 4 using both real and synthetic data sets, and the paper is concluded in Section 5 with an outlook to future work.

2. RELATED WORK

The classic probabilistic record linkage approach, as formalised in the 1960s by [15], has in recent years been improved by applying the expectation-maximisation (EM) algorithm for better parameter estimation in record pair classification [23], and by using approximate string comparisons to calculate partial agreement weights when record attribute (field) values have typographical variations [14, 25].

Since the mid 1990s, researchers have investigated a variety of approaches to record linkage, originating from artificial intelligence, database technology, information retrieval, machine learning, and data mining [9, 14, 25], with the objectives of improving the linkage quality and the scalability of the linkage process. Many of these approaches are based on supervised learning techniques and require training data (record pairs with known true match or true non-match status). Such training examples, however, are often not available in real world situations, or they have to be prepared manually. This is an expensive and time consuming process that can be quite error prone, as even humans sometimes are not able to clearly determine weather two records are a true match or not without having access to further information.

One supervised approach is to learn similarity measures for approximate string comparisons, such as the costs for edit-distance [3, 11] operations, with the aim to adapt similarity calculations to a particular data domain. Decision tree induction [13, 22] and support vector machines [19] are two popular supervised machine learning techniques that have been employed successfully for record pair classification. As expected, these techniques usually achieve better linkage quality compared to unsupervised approaches.

Three methods for record pair classification have been implemented in TAILOR [13]: the first is based on supervised decision tree induction, the second is using unsupervised k-means clustering (with three clusters, one each for matches, possible matches and non-matches), and the third is a hybrid approach that combines the first two to overcome the problem of lack of training data. It first clusters a sub-set of the weight vectors (again into matches, possible matches and non-matches), and then uses the match and non-match clusters to train a supervised decision tree classifier. Both the fully supervised and hybrid approach outperformed k-means clustering in experimental studies. In Section 4, a variation of the hybrid approach (employing an SVM instead of a decision tree classifier) will be compared experimentally to the proposed two-step classification approach.

Active learning is an approach that aims to overcome the problem of lack of training data. A system that presents a difficult to classify record pair to a user for manual classification is discussed in [21]. After such a pair has been manually classified by the user, it is added to the training data and the classifier is re-trained. This process is repeated until all record pairs are successfully classified. Using this approach, manually classifying less than 100 training pairs provided better results than a fully supervised approach that required 7,000 randomly selected examples. A similar approach is presented in [22], where a committee of decision trees is used to learn a set of rules that describe linkages.

Unsupervised clustering techniques have been investigated both to improve blocking [1, 12] and for automatic record pair classification [13, 16, 17]. The clustering techniques k-means and farthest-first were compared in [16] with supervised decision tree induction using both synthetic and real data. Surprisingly, farthest-first clustering outperformed k-means and achieved results comparable to decision trees. In [17], the k-means clustering algorithm has been employed to group weight vectors into matches and non-matches. In this approach, a user can also identify a 'fuzzy' region halfway in between the two cluster centroids where the difficult to classify record pairs are located. These pairs will then be handed to the user for manual clerical review. Using synthetic data, it was shown that this approach can significantly reduce the number of record pairs that have to be reviewed manually, while keeping high linkage quality.

In the past few years, unsupervised techniques based on relational clustering [2] have been explored in the area of entity resolution of relational data. While traditional record linkage techniques assume that only similarities between attribute values are available, in relational data the entities have additional relational information that can be used to improve the quality of entity resolution. Relational information can be present, for example, in census databases that include a family relationship attribute (with values such as 'married to', 'dependent of', or 'parent of'); or in bibliographic data where, besides the name of a paper, a publication record also contains a list of one or more authors that can indicate co-author relationships. Experimental results [2] showed that relational entity resolution outperforms non-relational entity resolution based only on record attribute similarities. However, non-relational data is still available in many real world situations, such as in databases that contain patient or customer information, and this paper concentrates on improving unsupervised classification of such non-relational data.

The two-step approach to record pair classification presented in this paper has been inspired by similar methods for text classification, where commonly only a limited number of positive labeled examples, besides many unlabeled examples, are available for training. In such situations the aim is to learn a classifier from these positive and unlabeled examples. In [26, 27], the PEBL and TC-WON approaches are presented, which are both based on iteratively training a SVM using the positive and a selected set of strong negative

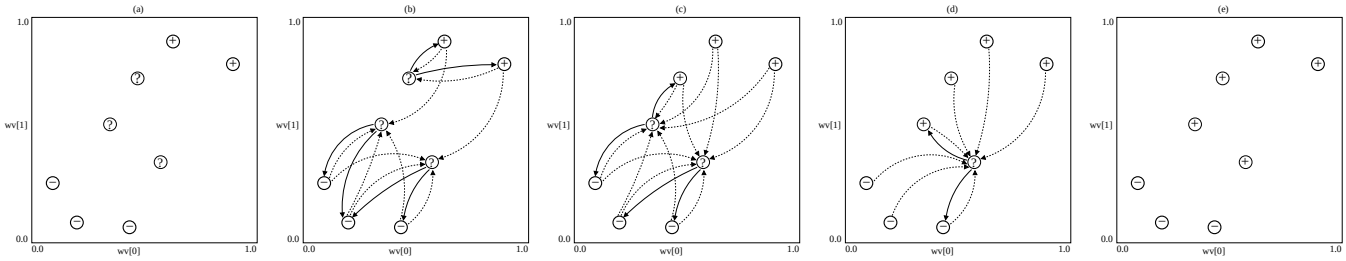

Figure 3: Example of the seeded nearest-neighbour classification process with 2-dimensional weight vectors and $k = 1$. Weight vectors classified as matches are shown with a plus, non-matches with a minus, and unclassified weight vectors with a question mark. The seed training examples are shown as bold circles. In each step, the unclassified weight vector closest to k already classified neighbours is added to one of the training sets. Details of this process are described in Section 3.2.1.

examples. Additional unlabeled examples are included into the negative training set as the trained classifier becomes more accurate, until all unlabeled examples are classified.

3. TWO-STEP CLASSIFICATION

The idea of seeded record pair classification is based on the following two assumptions. First, weight vectors that contain exact or high similarity values in all their matching weights were with high likelihood generated when two records that refer to the same entity were compared. Second, weight vectors that contain mostly low similarity values were with high likelihood generated when two records that refer to different entities were compared. As a result, selecting such weight vectors in a first step as seeds for generating training data, and training a classifier using these seed training examples in a second step, should enable automatic, efficient and accurate record pair classification.

Previously, in [6] and [7], the author has shown the feasibility of this proposed approach and investigated several variations of how to select the initial seed training examples. This paper concentrates on the second step of the approach, which will be discussed in detail in Section 3.2. First, an overview of the first step of the approach is given.

3.1 Step 1: Selection of Training Examples

The aim of the first step of the proposed classification approach is to select weight vectors from the set $\mathbf{W}$ of all weight vectors, generated in the comparison step, that with very high likelihood correspond to true matches and true non-matches, and to insert them into the match seed training examples set, $\mathbf{W}_M$, and the non-match seed training examples set, $\mathbf{W}_N$, respectively (with $\mathbf{W}_M \cap \mathbf{W}_N = \emptyset$). There are two main approaches to selecting training examples, either using distance thresholds or nearest-based.

The threshold based approach selects weight vectors that have all their matching weights within a certain distance threshold to the exact similarity or total dissimilarity values, respectively. For example, using the weight vectors from Figure 2 and a distance threshold of 0.2, only *WV(R1,R2)* will be selected into $\mathbf{W}_M$, and *WV(R1,R3)* and *WV(R2,R3)* into $\mathbf{W}_N$. The remaining three weight vectors will not be selected, as at least one of their matching weights is further than the 0.2 distance threshold away from 0 or 1.

In the nearest based approach, on the other hand, weight vectors are sorted according to their distances (using for example Manhattan or Euclidean distance) from the vectors containing only exact similarities, and only total dissimilarities, respectively, and the respectively nearest vectors are selected into the training sets. In Figure 2, *WV(R1,R2)* is closest to the exact similarities vector ([1.0, 1.0, 1.0, 1.0, 1.0]), followed by *WV(R3,R4)*; while *WV(R1,R3)* and *WV(R2,R3)* only contain total dissimilarity values, and *WV(R1,R4)* and *WV(R2,R4)* are the next vectors closest to them.

Experiments [6] showed that the nearest based approach generally outperforms threshold based selection. One reason is that nearest based selection allows explicit specification of the number of weight vectors to be included into $\mathbf{W}_M$ and $\mathbf{W}_N$. As weight vector classification is commonly a very imbalanced problem, the number of true non-matches in $\mathbf{W}$ is often much larger than the number of true matches [9], and thus more weight vectors should be selected into $\mathbf{W}_N$ than into $\mathbf{W}_M$. An estimation of the ratio r of true matches to true non-matches can be calculated using the number of records in the two databases to be linked, $\mathbf{A}$ and $\mathbf{B}$, and the number of weight vectors in $\mathbf{W}$:

$$r = \frac{min(|\mathbf{A}|, |\mathbf{B}|)}{|\mathbf{W}| - min(|\mathbf{A}|, |\mathbf{B}|)}, \quad (1)$$

with $|\cdot|$ denoting the number of elements in a set or a database. The problem with balanced training set sizes is that weight vectors that likely do not refer to true matches will be selected into $\mathbf{W}_M$ [6].

3.2 Step 2: Classification of Record Pairs

Once the seed training example sets for matches, $\mathbf{W}_M$, and non-matches, $\mathbf{W}_N$, have been generated, they can be used to train any binary classifier. In this paper, a nearest-neighbour based classifier and an iterative SVM classifier are investigated. In the following two sections, the set of weight vectors not selected into the seed training example sets is denoted with $\mathbf{W}_U$, with $\mathbf{W}_U = \mathbf{W} \setminus (\mathbf{W}_M \cup \mathbf{W}_N)$.

3.2.1 Nearest-Neighbour Classification

The basic idea of this classifier is to iteratively add unclassified weight vectors from $\mathbf{W}_U$ into the training sets until all weight vectors are classified. In each iteration, the unclassified weight vector closest to k already classified weight vectors is classified according to a majority vote of its classified neighbours (i.e. if the majority is either matches or non-matches). Using the seed training example sets, this nearest-neighbour based classifier can be implemented efficiently as illustrated in Figure 3 and detailed in Algorithm 1.

Algorithm 1: ***Seeded k-NN classification***

Input:
- Complete weight vector set: $\mathbf{W}$
- Seed training examples match set: $\mathbf{W}_M$
- Seed training examples non-match set: $\mathbf{W}_N$
- Distance function: $dist$
- Number of nearest-neighbours to consider: k

Output:
- Weight vectors classified as matches: $\mathbf{Z}_M$
- Weight vectors classified as non-matches: $\mathbf{Z}_N$

1: $\mathbf{Z}_M := \mathbf{W}_M$ and $\mathbf{Z}_N := \mathbf{W}_N$
2: $\mathbf{W}_T := (\mathbf{W}_M \cup \mathbf{W}_N)$ and $\mathbf{W}_U := \mathbf{W} \setminus \mathbf{W}_T$
3: Initialise empty heap $\mathbf{H}$
4: $\mathbf{M} := [], \mathbf{N} := [], \mathbf{U} := []$
5: **for** $(w_m \in \mathbf{W}_M)$ **do:**
6: $\quad \mathbf{M}[w_m] := \{(k+1)$ nearest $w_u \in \mathbf{W}_U$, sorted according to $dist(w_m, w_u)\}$
7: **end for**
8: **for** $(w_n \in \mathbf{W}_N)$ **do:**
9: $\quad \mathbf{N}[w_n] := \{(k+1)$ nearest $w_u \in \mathbf{W}_U$, sorted according to $dist(w_n, w_u)\}$
10: **end for**
11: **for** $(w_u \in \mathbf{W}_U)$ **do:**
12: $\quad \mathbf{U}[w_u] := \{(k+1)$ nearest $w_t \in \mathbf{W}_T$, sorted according to $dist(w_u, w_t)\}$
13: $\quad s := \sum_{w_t \in \mathbf{U}[w_u][1:k]} dist(w_u, w_t)$
14: $\quad$ Insert (s, w_u) into $\mathbf{H}$
15: **end for**
16: **while** $(\mathbf{W}_U \neq \emptyset)$ **do:**
17: $\quad (s, w_t) :=$ first element in $\mathbf{H}$
18: $\quad \mathbf{W}_U := \mathbf{W}_U - w_t$
19: $\quad$ **if** ($\mathbf{U}[w_t]$ contains more weight vectors from $\mathbf{Z}_M$ than $\mathbf{Z}_N$) **then:**
20: $\quad\quad \mathbf{Z}_M := \mathbf{Z}_M + w_t$
21: $\quad$ **else:**
22: $\quad\quad \mathbf{Z}_N := \mathbf{Z}_N + w_t$
23: $\quad$ **end if**
24: $\quad \mathbf{X}_u := \cup_{w_u \in \mathbf{U}[w_t]}(\mathbf{M}[w_u] \cup \mathbf{N}[w_u])$
25: $\quad$ **for** $(w_u \in \mathbf{X}_u)$ **do:**
26: $\quad\quad d := dist(w_t, w_u)$
27: $\quad\quad$ **if** (w_u is one of $(k+1)$ nearest to w_t) **then:**
28: $\quad\quad\quad$ Update w_t in $\mathbf{U}[w_u]$ with d
29: $\quad\quad\quad s := \sum_{w_v \in \mathbf{U}[w_u][1:k]} dist(w_u, w_v)$
30: $\quad\quad\quad$ Update (s, w_u) in $\mathbf{H}$
31: $\quad\quad$ **end if**
32: $\quad$ **end for**
33: $\quad$ **if** $(w_t \in \mathbf{Z}_M)$ **then:**
34: $\quad\quad \mathbf{M}[w_t] := \{(k+1)$ nearest $w_u \in \mathbf{X}_u$, sorted according to $dist(w_t, w_u)\}$
35: $\quad$ **else:**
36: $\quad\quad \mathbf{U}[w_t] := \{(k+1)$ nearest $w_u \in \mathbf{X}_u$, sorted according to $dist(w_t, w_u)\}$
37: $\quad$ **end if**
38: **end while**

In Algorithm 1, the number of nearest weight vectors to be considered when nearest neighbours are selected is denoted with k, with $k \geq 1$. The function $dist$ calculates the distance between two weight vectors, and can be any distance function such as Euclidean, Manhattan, or Cosine distance. In the first line of the algorithm, the output sets of classified match and non-match weight vectors, $\mathbf{Z}_M$ and $\mathbf{Z}_N$, are initialised to the seed training example sets. Next, in line 2, the sets $\mathbf{W}_T$ of all seed training examples, and $\mathbf{W}_U$ of all unclassified weight vectors, are generated. An empty heap data structure, $\mathbf{H}$, is then initialised next. A heap has the property that its first element is always the smallest element. It will be used in the second phase of the algorithm to iteratively get the next unclassified weight vector that has the smallest distance to the training sets. In line 4, three lists, $\mathbf{M}$, $\mathbf{N}$ and $\mathbf{U}$, are initialised that will be used to store nearest weight vectors as detailed below.

Lines 5 to 15 constitute the first phase of Algorithm 1. In lines 5 and 6, for each weight vector in the match seed training set $\mathbf{W}_M$, the closest $(k+1)$ not classified weight vectors from $\mathbf{W}_U$ are stored in the list $\mathbf{M}$. The same is done in lines 8 and 9 for the weight vectors in the non-match seed training set $\mathbf{W}_N$, with nearest neighbours from $\mathbf{W}_U$ stored in list $\mathbf{N}$. These $(k+1)$ nearest neighbours (in $\mathbf{M}$ and $\mathbf{N}$) are represented in Figure 3 using dashed arrowed lines. In lines 11 and 12, for each unclassified weight vector in $\mathbf{W}_U$, the closest $(k+1)$ weight vectors from the overall seed training set $\mathbf{W}_T$ are stored in the list $\mathbf{U}$. These nearest neighbours (in $\mathbf{U}$) are represented in Figure 3 using black arrowed lines (with the nearest neighbour indicated using a bolder black line). Additionally, in lines 13 and 14, the sum of the distances, s, of the k closest training set neighbours for each w_u in $\mathbf{W}_u$ are calculated and inserted into the heap $\mathbf{H}$. Therefore, at the end of this first phase of the algorithm, the first element of $\mathbf{H}$ will be the weight vector from $\mathbf{W}_U$ with the smallest distance sum to vectors from $\mathbf{W}_T$.

Phase two of Algorithm 1 (line 16 onwards) iterates until all weight vectors in $\mathbf{W}_U$ are classified. In line 17, the first element in $\mathbf{H}$, i.e. the weight vector with the smallest sum of distances to classified weight vectors, is taken from $\mathbf{H}$ and removed from $\mathbf{W}_U$ in line 18. Depending upon if the majority of neighbours of w_t are matches or non-matches, it is added to the set of classified matches, $\mathbf{Z}_M$, or classified non-matches, $\mathbf{Z}_N$, respectively (lines 19 to 22).

In line 24, the set $\mathbf{U}[w_t]$ of nearest training set weight vectors of w_t is used to create the set of nearest unclassified weight vectors, $\mathbf{X}_u$, retrieved via the corresponding nearest sets in the lists $\mathbf{M}$ and $\mathbf{N}$. Line 25 then loops over each of the unclassified weight vectors w_u in $\mathbf{X}_u$ that are nearest to w_t. In line 26, the distance d from w_u to the newly classified weight vector w_t is calculated, and the list of nearest classified weight vectors for w_u, $\mathbf{U}[w_u]$, is updated with this new distance d in line 28 if d is one of the $(k+1)$ smallest distances. In this case, the heap element for w_u also needs to be updated in $\mathbf{H}$ with the newly calculated distance sum s (lines 29 and 30). Finally, depending upon if w_t was classified as a match or a non-match, the set of nearest unclassified weight vectors for w_t is updated in the corresponding list $\mathbf{M}$ or $\mathbf{N}$ in lines 33 to 36.

This process is illustrated in Figure 3 (b) and (c). The middle upper, unclassified weight vector is classified as a match as its nearest neighbour is a match. Its list of nearest matches (solid lines) is used to get its nearest classified neighbours (the two seed matches in the top right), which in turn have lists of their nearest unclassified neighbours (dotted lines). The union of these lists becomes the new list of unclassified nearest neighbours, represented in Figure 3 (c) with the new dotted lines that point from the newly classified match to its two unclassified neighbours.

While naïve nearest-neighbour based classification would involve calculating the distances between all pairs of weight vectors, the seeded training sets allow a reduction through the use of the nearest lists **M**, **N** and **U**. In the first phase of Algorithm 1, distances are only calculated between the weight vectors in the overall training example set $\mathbf{W}_T$ and those in $\mathbf{W}_U$. If a fraction of t ($t < 1$) of all weight vectors is included in $\mathbf{W}_T$, then the number of distance calculations in phase one of the algorithm is $|\mathbf{W}|^2 \times (t - t^2)$. The maximum number of distance calculations will have to be done if half the weight vectors are in $\mathbf{W}_T$ and half are in $\mathbf{W}_U$, i.e. $t = 0.5$: $|\mathbf{W}|^2 \times 0.25$; while with $t = 0.1$, for example, the number of distance calculations to be done is only $|\mathbf{W}|^2 \times 0.09$. The second phase of the algorithm involves calculating a maximum of $(k+1) \times k$ distances for each of the weight vectors in $\mathbf{W}_U$, as for each of the k nearest training set weight vectors the $(k+1)$ nearest vectors will have to be checked for closeness.

The overall efficiency and scalability of the proposed nearest neighbour classifier can further be improved using data reduction or fast searching and indexing techniques as described in [20]. This is left for future work.

3.2.2 Iterative SVM Classification

The iterative SVM classifier is similar to the PEBL [26] and TC-WON [27] approaches for text and Web page classification based on only positive labeled training examples. The basic idea is to train an initial SVM using the seed training example sets $\mathbf{W}_M$ and $\mathbf{W}_N$, and to then iteratively add the strongest positive and negative classified weight vectors from $\mathbf{W}_U$ into the training sets of subsequent SVMs.

Algorithm 2 details the steps involved in this approach. The input parameter ip determines what percentage of unclassified weight vectors will be added into the training sets in each iteration, and tp determines the total percentage of weight vectors that will be added into the training sets. For example, if $tp = 100\%$ then all weight vectors from $\mathbf{W}$ will be used in the last iteration to train the final SVM.

The algorithm starts with initialising the training sets $\mathbf{T}_M$ for matches and $\mathbf{T}_N$ for non-matches, and by creating the set $\mathbf{W}_U$ of all unclassified weight vectors. The initial SVM is trained in line 3 using $\mathbf{T}_M$ and $\mathbf{T}_N$. The main loop then starts in line 5 and iterates until tp percent of all weight vectors have been included into the training sets.

Each iteration starts in line 6 by classifying the weight vectors in $\mathbf{W}_U$ using the previously trained SVM svm_i. The function $svm\_classify$ returns two sets, $\mathbf{X}_M$ and $\mathbf{X}_N$, that contain the weight vectors from $\mathbf{W}_U$ classified into matches and non-matches. These classified weight vectors are sorted in line 7 according to how far away they are from the SVM decision boundary, and in lines 8 and 9 the strongest positive and negative weight vectors are extracted into the sets $\mathbf{Y}_M$ of new matches, and $\mathbf{Y}_N$ of new non-matches. The size of these sets is determined by the increment percentage parameter ip. For example, if $ip = 50\%$, then in each iteration half of the weight vectors in $\mathbf{X}_M$ are inserted into $\mathbf{Y}_M$ and half of $\mathbf{X}_N$ into $\mathbf{Y}_N$. In lines 10 and 11, the new training examples in $\mathbf{Y}_M$ and $\mathbf{Y}_N$ are added to the training sets $\mathbf{T}_M$ and $\mathbf{T}_N$, and a new SVM svm_i is trained next in line 13 using these expanded training sets. Finally, in the last step within the iteration, in line 14, the new training examples from $\mathbf{Y}_M$ and $\mathbf{Y}_N$ are removed from the set $\mathbf{W}_U$ of unclassified weight vectors.

Algorithm 2: ***Seeded iterative SVM classification***

Input:
- Complete weight vector set: $\mathbf{W}$
- Seed training examples match set: $\mathbf{W}_M$
- Seed training examples non-match set: $\mathbf{W}_N$
- Increment percentage: ip
- Total training percentage: tp

Output:
- Weight vectors classified as matches: $\mathbf{Z}_M$
- Weight vectors classified as non-matches: $\mathbf{Z}_N$

1: $\mathbf{T}_M := \mathbf{W}_M$ and $\mathbf{T}_N := \mathbf{W}_N$
2: $\mathbf{W}_U := \mathbf{W} \setminus (\mathbf{W}_M \cup \mathbf{W}_N)$
3: $svm_0 := train\_svm(\mathbf{T}_M, \mathbf{T}_N)$
4: $i := 0$
5: **while** $(|\mathbf{T}_M| + |\mathbf{T}_N|) < (|\mathbf{W}| * tp/100)$ **do:**
6: $\mathbf{X}_M, \mathbf{X}_N := svm\_classify(svm_i, \mathbf{W}_U)$
7: Sort $\mathbf{X}_M$ and $\mathbf{X}_N$ according to distance from svm_i decision boundary (hyperplane)
8: $\mathbf{Y}_M := |\mathbf{X}_M| * (ip/100)$ vectors from $\mathbf{X}_M$ furthest away from decision boundary
9: $\mathbf{Y}_N := |\mathbf{X}_N| * (ip/100)$ vectors from $\mathbf{X}_N$ furthest away from decision boundary
10: $\mathbf{T}_M := \mathbf{T}_M \cup \mathbf{Y}_M$
11: $\mathbf{T}_N := \mathbf{T}_N \cup \mathbf{Y}_N$
12: $i := i + 1$
13: $svm_i := train\_svm(\mathbf{T}_M, \mathbf{T}_N)$
14: $\mathbf{W}_U := \mathbf{W}_U \setminus (\mathbf{Y}_M \cup \mathbf{Y}_N)$
15: **end while**
16: $\mathbf{X}_M, \mathbf{X}_N := svm\_classiy(svm_i, \mathbf{W}_U)$
17: $\mathbf{Z}_M := \mathbf{T}_M \cup \mathbf{X}_M$ and $\mathbf{Z}_N := \mathbf{T}_N \cup \mathbf{X}_N$

The final SVM is then used in line 16 to classify the weight vectors in $\mathbf{W}_U$ that have not been classified so far (this step is not required if $tp = 100\%$), and in line 17 the final two sets of matches, $\mathbf{Z}_M$, and non-matches, $\mathbf{Z}_N$, are created.

Assuming that training a SVM is of quadratic complexity in the number of training examples [27], then the overall complexity of Algorithm 2 is $O(|\mathbf{W}| * i)$, with i being the number of times a SVM is trained. The value of i depends upon the ip and tp parameter values. For example, if $ip = 50\%$ and $tp = 100\%$ then $i = log_2(|\mathbf{W}_U|)$, while if $ip = 25\%$ and $tp = 50\%$ then $i = 3$, as in each step 25% of weight vectors from $\mathbf{W}_U$ will be added to the training sets. Training of SVMs will become increasingly time consuming as more weight vectors are added into the training sets.

4. EXPERIMENTAL EVALUATION

The two record pair classifiers presented above were evaluated and compared with two other classification methods. The first is a fully supervised SVM that has access to the true match status of all weight vectors. Nine parameter variations were evaluated: three kernel methods (linear, polynomial and RBF), and three values for the cost parameter, C [4] (0.1, 1, 10). The second method is based on the hybrid approach implemented in the TAILOR [13] toolbox. It first employs k-means (with one cluster each for matches, possible matches and non-matches), and then uses the match and non-match clusters to train a SVM (in [13] a decision tree classifier has been used instead). Two distance functions (Manhattan and Euclidean) were evaluated for the k-means step, while for the SVM classifier step the same nine parameter variations as for the supervised SVM were used.

Table 1: Data sets used in experiments. See Section 4 for more details.

Data set	Number of records	Task	Pairs completeness	Reduction ratio	Number of weight vectors (i.e. $\|\mathbf{W}\|$)	Ratio r according to Equation (1)
Census	449 + 392	Linkage	1.000	0.988	2,093	1 / 4.34
Restaurant	864	Deduplication	1.000	0.713	106,875	1 / 122.7
Cora	1,295	Deduplication	0.924	0.793	173,769	1 / 133.2
DS-Gen-A	1,000	Deduplication	0.957	0.995	2,475	1 / 1.48
DS-Gen-B	2,500	Deduplication	0.940	0.997	9,878	1 / 2.95
DS-Gen-C	5,000	Deduplication	0.953	0.997	35,491	1 / 6.10
DS-Gen-D	10,000	Deduplication	0.948	0.997	132,532	1 / 12.25

For the two-step classification approach, the imbalanced nearest based selection was used, with the number of seed training examples in $\mathbf{W}_N$ selected as 5% or 10% of all weight vectors in $\mathbf{W}$, respectively, and the number of training examples in $\mathbf{W}_M$ was calculated according to the ratio r as given in Equation (1). For the nearest-neighbour based two-step classifier, again Manhattan and Euclidean distances were evaluated in combination with k set to 3 and 9. For the iterative SVM based two-step classifier, the same nine parameter variations as for the supervised SVM were evaluated, and the parameters *ip* and *tp* were set to the pairs (0,0) (no iterative refinement), (25,25), (25,50), and (50,100).

The experiments for the supervised SVM and TAILOR classifiers were conducted using 10-fold cross validation (90% used for training and 10% for testing), while this was not possible for the two-step classifier because the selection of seed training examples requires all weight vectors in $\mathbf{W}$.

All classifiers are implemented in the *Febrl* [8] record linkage system, which is written in Python. The *libsvm* library was used for the SVM classifier [4]. All experiments were run on a 2.13 GHz dual-core CPU with 2 GBytes of main memory, running Linux 2.6.20 and using Python 2.5.1.

Experiments were conducted using both real and synthetic data, as summarised in Table 1. Three real data sets from the *SecondString* toolkit[1] were used, while four synthetic data sets of various sizes were created using the *Febrl* data set generator [5]. This synthetic data contains name and address attributes that are based on real-world frequency tables, and includes 60% original and 40% duplicate records. These duplicates were randomly created through modification of record attributes (like inserting, deleting or substituting characters, and swapping, removing, inserting, splitting or merging words), again according to real-world error characteristics. Up to nine duplicates were generated for one original record, with a maximum of three modifications per attribute and a maximum ten modifications per record.

Standard blocking [1] was applied in all experiments to reduce the number of record pair comparisons, with the blocking keys being combinations of name, address and postcode values. In the record pair comparison step, the Winkler [24] approximate string comparator (commonly used in record linkage) was employed for name, address, paper title and conference name attribute (field) comparisons. Additionally, character difference comparisons [8] were used on attributes such as postcode, street number, and publication year.

The quality of the compared record pairs is shown in Table 1 using the *pairs completeness* measure (which is the number of true matched record pairs generated by blocking divided by the total number of true matched record pairs), and the complexity of the record pair comparison step is shown using the *reduction ratio* measure (which is one minus the number of record pairs generated by blocking divided by the total possible number of record pairs) [9, 13].

Due to the normally imbalanced distribution of matches and non-matches in the weight vector set $\mathbf{W}$, the accuracy measure commonly used for evaluating classifier performance is not suitable for assessing the quality of record pair classification [9]: the large number of non-matches would dominate accuracy, and show results that are too optimistic. Instead, the F-measure, F, the harmonic mean of precision, P, and recall, R, is used for measuring classifier quality: $F = 2(P \times R)/(P + R)$, with $P = TP/(TP + FP)$ and $R = TP/(TP + FN)$. TP is the number of true positives (true matched record pairs classified as matches), FN the number of false negatives (true matched record pairs classified as non-matches), and FP the number of false positives (true non-matched record pairs classified as matches).

Table 2 shows the quality of the seed training example sets generated in the first step of the proposed two-step classification approach, given as the percentage of correctly selected weight vectors in these sets, i.e. |true matches in $\mathbf{W}_M$|/|$\mathbf{W}_M$| and |true non-matches in $\mathbf{W}_N$|/|$\mathbf{W}_N$|. For the second step, Figure 4 shows the average F-measure results, together with the minimum and maximum F-measure values over all parameter variations discussed above (i.e. 9 supervised SVMs, 18 TAILOR classifiers, 8 nearest-neighbour based two-step classifiers, and 72 iterative two-step SVM classifiers – 18 each for the four variations of the (*ip*,*tp*) parameter pairs). These average results, rather than the 'best' results using a certain parameter setting, are presented as they provide a more realistic picture of the overall performance of a classifier (which likely depends upon the characteristics of a data set).

4.1 Results and Discussion

As can be seen in Table 2, the seed training example sets selected in the first step of the proposed two-step classification approach are mostly of very high quality, with the match training example set $\mathbf{W}_M$ only containing true matches in all but one case (10% seed size for the 'Restaurant' data set). While overall the 1% training set selection contains the highest quality seed training data, the size of these sets (especially $\mathbf{W}_M$) is very small, and the resulting classifiers based on these 1% seeds generated in the second step were much worse in most experiments compared to the classifiers generated based on 5% or 10% seed sizes. Therefore, the 1% seed size classifiers were not included into the F-measure results presented in Figure 4.

[1] `http://secondstring.sourceforge.net`

Table 2: Quality of nearest-based seed training example selection as described in Section 3.1. Each pair of values shows the quality of $\mathbf{W}_M$ / $\mathbf{W}_N$ as percentage of correctly selected training examples. The seed size gives the percentage of weight vectors from $\mathbf{W}$ selected into $\mathbf{W}_N$.

Seed size	Census	Restaurant	Cora	DS-Gen-A	DS-Gen-B	DS-Gen-C	DS-Gen-D
1%	100%/100%	100%/100%	100%/96.8%	100%/100%	100%/99.0%	100%/100%	100%/100%
5%	100%/100%	100%/100%	100%/98.0%	100%/96.7%	100%/98.4%	100%/99.8%	100%/99.8%
10%	100%/100%	90.8%/100%	100%/97.6%	100%/95.5%	100%/98.3%	100%/99.5%	100%/99.7%

The F-measure results for six data sets are shown in Figure 4 (the results for the small 'DS-Gen-A' data set are similar to those of the other synthetic data sets). As can be seen, there is a large range of F-measure result values for certain classifiers, and very different F-measure values for the same classifier on the different data sets.

As expected, the supervised SVM classifier performs best on all data sets. For the 'Census' and 'Restaurant' data sets, there are however parameter settings that lead to SVM classifiers that perform worse than the best two-step classifier using the iterative SVM approach. The TAILOR hybrid classifier approach has the lowest performance on most data sets. Only on the 'Cora' data set it achieves better results than most two-step classifier variations.

The nearest-neighbour based two-step classifier performs better than all iterative SVM variations for all synthetic data sets, while for the real data sets the iterative SVM generally achieves better classification results. Looking at the different values of the parameter pairs (ip,tp), a noticeable improvement when including more weight vectors into the training sets is only visible for the 'Cora' data set, while the improvements are very small for the three smaller synthetic data sets, and mixed for the 'DS-Gen-D' and 'Census' data sets. For the 'Restaurant' data set, the results are even getting worse when more training data is added.

The results for the 'Restaurant' data set are generally very low for all unsupervised classifiers compared to the supervised SVM. The reasons for this are that for this data set the weight vector set $\mathbf{W}$ only contains 112 true matches (duplicates), but 106,763 non-matches; and also because the attributes in this data set contain addresses with a large proportion of either abbreviations or completely different values (such as 'Los Angeles' versus 'Beverly Hills') for the same restaurant. Therefore, the weight vectors generated when such attribute values were compared have a very mixed distribution of matches and non-matches that are hard to classify without knowing the true match status of these weight vectors. This can already be seen in Table 2 where with the 10% seed size the match training set $\mathbf{W}_M$ contains more than 9% non-matches.

The experiments presented in this paper show that the proposed two-step classification approach can achieve results that outperform other unsupervised record pair classification techniques, such as the hybrid TAILOR approach which has shown to be better than k-means clustering [13]. On the other hand, these experiments also showed the limitations of unsupervised classification based on only pair-wise attribute similarities, compared to supervised classification.

5. CONCLUSIONS AND FUTURE WORK

This paper presented a novel unsupervised two-step approach to record pair classification that is aimed at automating the record linkage process. This approach combines automatic selection of seed training examples with training of a binary classifier. The two classifiers discussed (nearest-neighbour based and iterative refinement of a SVM) achieve improved record pair classification results compared to other unsupervised classifiers, such a the hybrid TAILOR [13] approach. Thus, the proposed approach can be used in situations where no training data is available.

Future work will include to conduct more experiments using different data sets, including run-time tests on data sets of various sizes in order to experimentally get scalability results. Related to this is the implementation of data reduction and fast searching and indexing techniques for the nearest-neighbour based classifier [20], and similar approaches for the iterative SVM, with the aim to reduce training times while keeping a high record pair classification quality. Another area of research will be to investigate active learning techniques [21, 22] and combine them with the seeded training example selection approach presented here. Active learning can, for example, be used in the iterative two-step SVM classifier to select the hardest to classify examples and hand them to a user for manual classification.

6. ACKNOWLEDGEMENTS

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

7. REFERENCES

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

[2] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), 2007.

[3] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *ACM KDD'03*, pages 39–48, Washington DC, 2003.

[4] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. Manual, Department of Computer Science, National Taiwan University, 2001. Software available at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] P. Christen. Probabilistic data generation for deduplication and data linkage. In *IDEAL'05, Springer LNCS 3578*, pages 109–116, Brisbane, 2005.

[6] P. Christen. A two-step classification approach to unsupervised record linkage. In *AusDM'07, CRPIT vol. 70*, pages 111–119, Gold Coast, Australia, 2007.

[7] P. Christen. Automatic training example selection for scalable unsupervised record linkage. In *Accepted for*

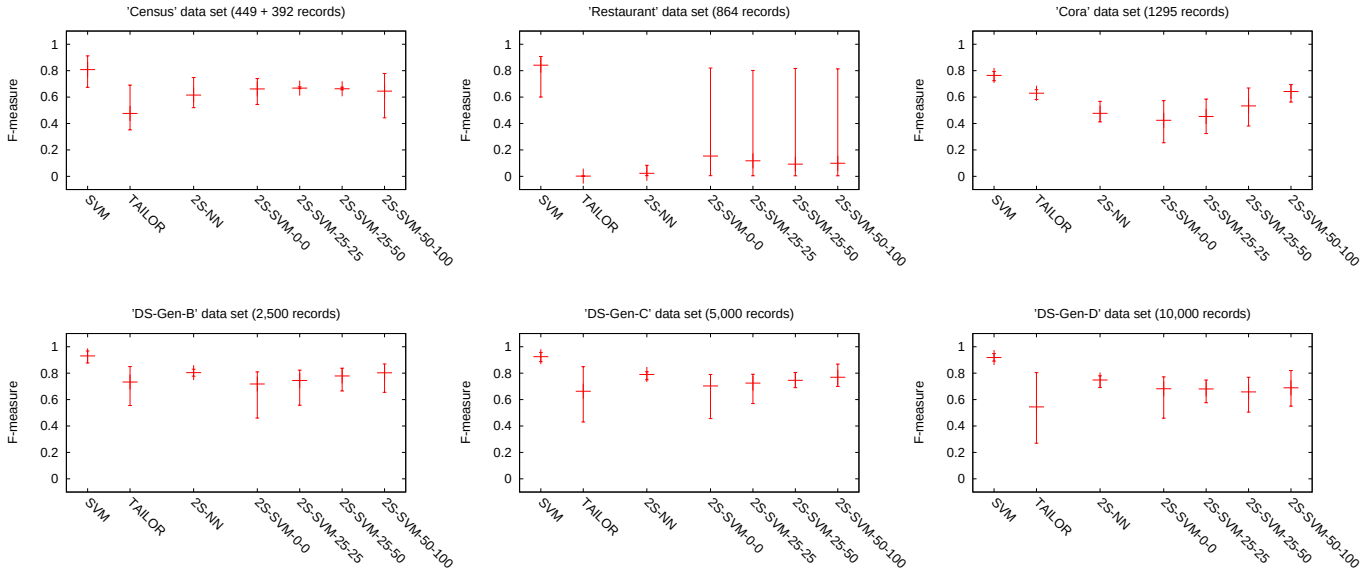

Figure 4: Average F-measure results (shown with minimum and maximum values).

PAKDD'08, Osaka, Japan, 2008.

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

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

[10] T. Churches, P. Christen, K. Lim, and J. X. Zhu. Preparation of name and address data for record linkage using hidden Markov models. *BioMed Central Medical Informatics and Decision Making*, 2(9), 2002.

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

[12] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *ACM KDD'02*, pages 475–480, Edmonton, 2002.

[13] M. Elfeky, V. Verykios, and A. Elmagarmid. TAILOR: A record linkage toolbox. In *ICDE'02*, pages 17–28, San Jose, 2002.

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

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

[16] K. Goiser and P. Christen. Towards automated record linkage. In *AusDM'06, CRPIT*, volume 61, pages 23–31, Sydney, 2006.

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

[18] J. Jonas and J. Harper. Effective counterterrorism and the limited role of predictive data mining. *Policy Analysis*, (584), 2006.

[19] U. Y. Nahm, M. Bilenko, and R. J. Mooney. Two approaches to handling noisy variation in text mining. In *TextML'02*, pages 18–27, Sydney, 2002.

[20] J. S. Sanchez, J. M. Sotoca, and F. Pla. Efficient nearest neighbor classification with data reduction and fast search algorithms. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4757–4762, 2004.

[21] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *ACM KDD'02*, pages 269–278, Edmonton, 2002.

[22] S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *ACM KDD'02*, pages 350–359, Edmonton, 2002.

[23] W. E. Winkler. Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. Technical Report RR2000/05, US Bureau of the Census, 2000.

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

[25] W. E. Winkler. Overview of record linkage and current research directions. Technical Report RR2006/02, US Bureau of the Census, 2006.

[26] H. Yu, J. Han, and K. C.-C. Chang. PEBL: Positive example based learning for Web page classification using SVM. In *ACM KDD'02*, pages 239–248, Edmonton, 2002.

[27] H. Yu, C. X. Zhai, and J. Han. Text classification from positive and unlabeled documents. In *CIKM'03*, pages 232–239, New Orleans, 2003.