Regression classifier for Improved Temporal Record Linkage

Yichen Hu **Qing Wang** **Dinusha Vatsalan** **Peter Christen**

Research School of Computer Science,
The Australian National University,
Canberra ACT 0200, Australia
Email: {yichen.hu,qing.wang,dinusha.vatsalan,peter.christen}@anu.edu.au

Abstract

Temporal record linkage is the process of identifying groups of records which are collected over long periods of time, such as census databases or voter registration databases, that represent the same real-world entities. These datasets often contain temporal information for each record, such as the time when a record was created, or the time when it was modified. Unlike traditional record linkage, which treats differences between records from the same entity as errors or variations, temporal record linkage aims to capture records from entities where the details of these entities change over the time.
This paper proposes a temporal record linkage approach that learns the probabilities for attribute values of records to change within different periods of time, which extends an existing temporal approach *decay model*. The proposed method uses a regression based machine learning model to predict decay with sets of attributes, where attribute values in each set could affect the decay of others. Our experimental results show that the proposed approach results in generally better recall than baseline approaches on real-world datasets.

Keywords: Data matching, entity resolution, record linkage, temporal data

1 Introduction

Record linkage (also known as data matching, entity resolution, and duplicate detection) identifies records that refer to the same real-world entity (Christen 2012*a*). Record linkage is being used in many application domains, such as linking patient data for disease outbreak detection or clinical trails in the health industry, credit checking and fraud detection in the finance industry, and constructing population databases for social science research (Kum et al. 2014). Challenges in record linkage are caused by the lack of unique identifiers (such as national identifier number), dirty data (such as misspellings and missing values), legitimate updates over time (such as changes in last name and address), and the lack of informative attributes (many datasets do not have gender and/or date of birth. e.g. no gender or date of birth in publication datasets).

Record linkage generally involves the following steps: data preprocessing (such as unifying data structure and cleansing datasets), blocking/indexing (grouping records into blocks, where records with a certain similarity are grouped into the same block), comparison and classification (comparing pairs of records in each block to decide if they are a match), and evaluation (Christen 2012*a*). This paper focuses on record pair comparison and classification, but we will also briefly discuss blocking/indexing.

Record linkage has been studied extensively in the past few decades. However, until recently, most works in this field did not use any temporal information available in datasets (Li et al. 2012). Records of the same entity can be collected over a long period of time (multiple years or even decades, such as census data in Australia collected every five years). During such periods the attribute values of an entity are likely to change, such as job position, living address, and potentially last name. Traditional record linkage methods often assume records that are highly similar are most likely to be belonged to the same entity. These techniques do not perform well on temporal records, because many entities have changed some of their attribute values over time. For example, when a person has changed his or her last name and address over a few years, their earlier records can be linked by mistake to records of a different person who has the same last name and/or address.

Temporal record linkage tries to address the above issues by using temporal information (such as the time-stamp when a record was created) from each record as a special type of attribute. These time-stamps can be used to sort records by time order, calculate time distance between records, and therefore provide potential for new record linkage approaches (examples to be discussed in Section 2). To be used in temporal record linkage, a dataset should contain temporal data for each record, such as date entered (for registration dataset), date being published (for publication datasets), and date being collected (for datasets collected by taking snapshots of databases at different times).

This paper extends an existing temporal linkage approach called *decay model* (Li et al. 2012). The *decay model* learns the probability for an attribute to change over time (*disagreement decay*), and the probability for an attribute to share the same value among different entities over time (*agreement decay*), and then uses these decays to compute and adjust the weight given to each attribute. The sum of the adjusted attribute weights is used to calculate the similarity between a pair of records and decide if they are a match or non-match based on a similarity threshold (Christen 2012*a*). We integrate a regression model (such as linear regression) into

the *decay model*. The regression model uses multiple support attributes to calculate the decay of a main attribute whose decay is affected by the values of those support attributes. The calculated decays are more reflective to each entity's specific situation. For example, a person's gender can affect the likelihood of changes in their last name, so when we calculate a decay of last name with gender as additional input, we can produce a gender sensitive decay for last name. We choose linear regression model because it is commonly used in parameter estimation and prediction (Krueger et al. 2015).

2 Related Works

We discuss related works in three areas: generic record linkage techniques, temporal models, and temporal linkage techniques.

Generic record linkage techniques: Generic record linkage techniques refer to the class of algorithms that do not consider temporal information explicitly. Record-wise or cluster-wise similarity comparison functions are sometimes treated as *black-box*. The similarities produced by these function are used to conduct linkage tasks.

Benjelloun et al. (2009) proposed three variations of the Swoosh algorithms (G-Swoosh, R-Swoosh, and F-Swoosh), which handle the record linkage problem by merging each pair of matched records into a new record. Merged records are removed and the newly created record is added to the input dataset. The matching and merging behaviors and criteria are defined by the user. We use F-Swoosh in our approach to link records.

Kim & Lee (2010) proposed an algorithm which uses Locality-sensitive hashing (LSH) (Indyk & Motwani 1998) to iteratively group records to clusters. LSH approximately compares the Jacarrd similarity between records, and place records that are similar to each other to the same cluster (so that any pair of records from that cluster will have a similarity higher than a certain threshold). Records placed into the same cluster are merged into a new record, and the algorithm conducts LSH again in an attempt to merge more records, until an user defined termination condition is met. We use a LSH based blocking technique which we will explain in Section 4.1.

Li et al. (2012) proposed a clustering technique, which computes the similarity between a record and a cluster of records. If a record has higher similarity to another cluster than its current one, this record will be reassigned to the new cluster. In the case where a record's similarity to any cluster is lower than a user-defined threshold, a new cluster is created with this record as its member. The adjustment process is repeated until the result converges or oscillates on the number of clusters.

Temporal models in record linkage: Temporal models are used to adjust attribute-wise or record-wise similarities with temporal information. A temporal model is often learned from a training dataset (which is the case for all of the approaches below). The learned model is then applied in attribute or record pair comparison process, to adjust the finial similarity score.

Li et al. (2012) proposed a similarity measure which considers the probability of an attribute's value to change over a certain time period (the probability is learned from training data). The algorithm calculates a disagreement rate (the probability for an attribute to change within a certain time interval) and an agreement rate (the probability for an entity's attribute value to be the same with a different entity within a time interval). The two rates are used to adjust the weight of each attribute.

More recently, Li et al. (2015) proposed a temporal model which learns the probability for each attribute value to be changed to another attribute value over a certain time period. However, this approach is restricted to attributes whose values might change, such as job positions (for example, the position 'technician' can be changed to 'manager').

Christen & Gayler (2013) adapted the approach by Li et al. (2012), which adjusts the temporal model iteratively using linkage results produced. The difference between this algorithm and the original one is that the original algorithm only learns the temporal model from training data, whereas this algorithm continuously trains the temporal model using linkage results produced by itself.

Chiang et al. (2014*a*) proposed an algorithm which learns the probability of an attribute's value to recur at different time intervals. For each value of an attribute, the algorithm constructs a transition history and uses the history to calculate the probability for a value to recur. These recurrence probabilities are used to adjust similarity weighting of record pairs to improve entity resolution quality.

Temporal linkage techniques: Unlike generic record linkage techniques, temporal linkage techniques use temporal information that is available in a dataset. Unlike temporal models, temporal linkage techniques do not adjust the way in which similarities are calculated between records, but they adjust record comparisons such as the order of comparisons between records, or between clusters of records.

Chiang et al. (2014*b*) proposed a clustering algorithm which processes records in two phases using different temporal models. In the first phase, the algorithm greedily groups records into clusters and creates temporal signatures for each of the clusters. In the second phase, the algorithm calculates the similarities between clusters, and adjusts these similarities using the temporal signatures, to decide if two clusters need to be merged.

Li et al. (2015) recently proposed a temporal clustering algorithm which identifies data-sources that are likely to be well-updated (accurately describe the current state of entities), and then uses these well-updated sources to create the initial clustering before using other data-sources that are not fresh.

3 Notation and Problem Statement

We now provide the notation we use in this paper and define the problem we aim to tackle.

Entity: Given a domain with a set of entities $\mathbf{E}$, where each entity $e \in \mathbf{E}$ is described by a set of attributes $\mathbf{A}$.

Record: Let $\mathbf{R}$ be a set of records, $r \in \mathbf{R}$ refers to a record with a time-stamp t. Each record r has a list of attribute values $[a_1, a_2, ..., a_k]$, where the value of an attribute $A \in \mathbf{A}$ in a record r is denoted as $r.A$. Every record $r \in \mathbf{R}$ must belong to exactly one entity $e \in \mathbf{E}$. The entity that associated with a record r is denoted as $r.e$.

Attribute values of an entity e can change over time, where each change (update) is represented by a record r with a time-stamp t and attribute value(s) that is different from the previous record. For example, let r_1, r_2 be two records belonging to e (in another word, $r_1.e = r_2.e$). If $r_1.t < r_2.t$ and

$\exists A \in \mathbf{A} : r_1.A \neq r_2.A$, then we say that the value of attribute A of entity e was changed between two time-stamps $r_1.t$ and $r_2.t$.

Training dataset: Given a training dataset $\mathbf{C}$ in the form of a set of clusters of records. Each cluster $C \in \mathbf{C}$ contains a set of records $\{r_1, r_2, ...\}$ that represents an entity e from the domain. All records in a cluster C represent the same entity, and all records referring to the same entity are in the same cluster C.

Problem statement: Temporal record linkage is the problem of grouping a set of records $\mathbf{R}$ into a set of clusters $\mathbf{C}'$. Ideally, for each created cluster $C' = \{r_1, r_2, ...\}$, and $C' \in \mathbf{C}'$, C' represents an entity $e_j \in \mathbf{E}$. All records in a cluster $C' \in \mathbf{C}'$ belong to the same entity: $\forall r \in C' \rightarrow r.e = e_j, e_j \in \mathbf{E}$. All records that are belonging to the same entity are in the same cluster: $\forall r_1 \in \mathbf{R}$, $\forall r_2 \in \mathbf{R}$, $r_1.e = r_2.e \leftrightarrow \exists C' \in \mathbf{C}'(r_1 \in C' \wedge r_2 \in C')$.

Our work addresses the temporal record linkage problem using a weighting strategy which adjusts the weight of each attribute. The objective of our work is to improve the quality of linkage. Such a weighting strategy is also called a temporal model, which is trained by a training dataset $\mathbf{C}$.

Temporal model training: Given a training dataset $\mathbf{C}$ with ground truth, the problem of training a temporal model is to build a statistical model for attribute weighting. This model can adjust the weight of each attribute A when a pair of records are being compared. The adjusted weights reflect the temporal characteristics of the whole dataset.

Our work addresses the temporal model training problem with a machine learning approach, by using a regression model to train and predict parameters for temporal model. Given an attribute A from a record r, and a time distance Δt, there exists a probability $p^{\neq}$ that A changes its current value within time distance Δt. Records used to train the model can thus be created with two class values: *changed* or *unchanged* using training data $\mathbf{C}$.

Training data generation: Given a cluster $C \in \mathbf{C}$ where $C = \{r_1, r_2, ...\}$. Given a record r_i in C and a time distance Δt, we can find all records $r' \in C - \{r_i\}$ where $r'.t - r_i.t \leq \Delta t$ holds. If there exists a record r' such that $r'.A \neq r_i.A$, a record for training can be created with class value *changed*, with Δt and attribute values of r_i as features. Similarly, if every record r' satisfies the condition: $r'.A = r_i.A$, a record for training with class value *unchanged* is created.

4 Framework

Figure 1 presents a high-level overview of our record linkage process. With a set of records $\mathbf{R}$ as input, a blocking method first places records into smaller sets, and only compares records within each set (there can be overlaps between these sets, i.e. each record can be inserted into more than one set). The reason for using a blocking method is to cluster records that are similar to each other into smaller sets, therefore improve the scalability and reduce computational cost for linkage approaches that have a running time growing exponentially by the size of dataset. The similarity threshold for threshold based blocking methods is usually defined by the user (Christen 2012*b*). The linkage technique computes the similarities between the records within those blocks (Christen 2012*a*). The criteria for two records (or clusters of records) to link are defined by specific linkage technique, such as by

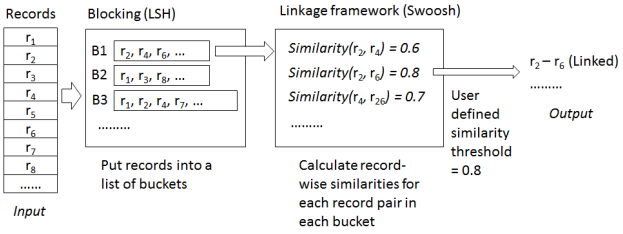


Figure 1: Overview of our linkage approach. Given a set of records as input, our blocking approach places the records in blocks (each record can be placed in multiple blocks). For each block, a linkage framework calculates similarities of record pairs and links records according to certain criteria, such as a user defined similarity threshold.

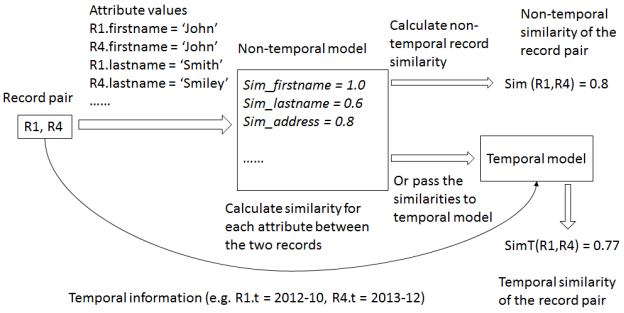


Figure 2: Temporal and non-temporal similarity values of a given pair of records. The difference is that a temporal model combines temporal information with attribute similarities to calculate the record-wise similarity.

a user-defined similarity threshold. In this paper, we use LSH (Locality Sensitive Hashing) (Indyk & Motwani 1998) for blocking, and F-Swoosh (Benjelloun et al. 2009) as linkage technique. Both will be further explained in detail in the following sections.

Figure 2 shows how a pair-wise record similarity is produced with and without a temporal model, as well as the problem domain this paper will address. The temporal model takes attribute-wise similarities of a record pair as input, as well as temporal information related to the record pair (such as the dates when the records were added to the dataset). Same as the a non-temporal model, the final output of a temporal model is a similarity score for a pair of records, where these similarity scores are adjusted by the temporal information.

4.1 Blocking: Locality-Sensitive Hashing

Blocking methods create smaller record sets based on the original dataset, where records in each block possibly refer to similar entities. Only those records within the same blocks are compared with each other. Conceptually, a blocking method can be understood as a linkage method with a low threshold which is only concerned with recall and not precision.

We use LSH to create blocks. LSH is an approximation algorithm used to cluster similar texts. The texts in the same cluster will have a high Jaccard similarity (computed by shingling a text into a set of q-grams) above a user defined threshold (Rajaraman & Ullman 2011). We use LSH as the blocking method based on the observation that a single attribute value

Algorithm 1 Blocking with LSH

Input:
- A set of records: $\mathbf{R}$
- A list of lists that contains attributes for blocking: $\mathbf{L}_A$
- A similarity threshold t_s
- Maximum block size $size_{max}$

Output:
- Sets of record ids, each set is a block: $\mathbf{B}$

```
1: B = set()
2: //For each list from L_A, produce a set of blocks
3: for l_A in L_A do
4:    //Create a hashtable where each
5:    //record-reference links to a list of block keys
6:    D_I = hashtable()
7:    //Generate block-keys for each record
8:    for r in R do
9:       gramset = set()
10:      for A in l_A do
11:         //Get the q-grams of the attribute value
12:         //q is defined by the user for each attribute
13:         for gram in GetQGrams(r.A, q) do
14:            gramset.add(gram)
15:      blockkeys = LSHBucketKeys(gramset, t_s)
16:      D_I[r.recordid] = blockkeys
17:   D_b = hashtable()
18:   //Group record-references by block-keys
19:   for recordid, blockkeys in D_I do
20:      for key in blockkeys do
21:         if key not in D_b then
22:            D_b[key] = set()
23:         D_b[key].add(recordid)
24:   //Remove over-sized blocks
25:   for block in D_b.values() do
26:      if length(block) >= size_max then
27:         D_b.remove(block)
28:   B = B ∪ D_b.values()
29: return B
```

(such as first name), or the concatenation of a list of attribute values (such as the string concatenation of first name, last name, and zip code), can be considered as a text string. If we consider a string value as a blocking key value of a record, we can put similar records into the same block by comparing their blocking key values using LSH. LSH is chosen as our blocking approach because it is scalable and efficient, performing reasonably good when comparing it to other blocking approaches (Wang et al. 2016).

After blocks of records are produced by LSH, we remove the over-sized blocks by a user defined maximum block size ($size_{max}$), to further reduce the computational cost. $size_{max}$ was introduced as we observed that some blocks can be very large due to commonly shared attribute values (such as first name 'David'). As a result, these large blocks significantly increased the overall run-time of the algorithm. A similar approach was used in suffix array based blocking (de Vries et al. 2011).

Algorithm 1 describes the way we use LSH for blocking in our work. From lines 8 to 16, a list of blocking keys is created for each record and hashed into the hashtable $\mathbf{D}_I$ by its record ID. The function $GetQGrams()$ takes a text input value and an integer q (Ukkonen 1992). The value of q decides the length of q-grams which will be created by shingling the text input into q-grams. The function $LSHBucketKeys()$ takes a set of q-grams and a similarity threshold to produce LSH buckets. We treat the $LSHBucketKeys()$ function as a blackbox in this work as its internal mechanism is dependent on the implementation of LSH. Each bucket key uniquely identifies a block. For example, if the hashing of a record results in ten keys, this record is hashed into ten blocks by LSH. From lines 19 to 23, $\mathbf{D}_I$ is converted to another hashtable $\mathbf{D}_b$ where each blocking key is mapped to a list of record IDs (each list is a block). In lines 25 to 27, the algorithm applies the $size_{max}$ parameter and removes over-sized blocks.

Algorithm 2 The Swoosh algorithm (conceptual)

Input:
- A set of records: $\mathbf{R}$

Output:
- A set of records, each record represents an entity: $\mathbf{E}$

```
1: E = ∅
2: while R ≠ ∅ do
3:    currentRecord = R.getFirstItem()
4:    R.removeFirstItem()
5:    buddy = null
6:    for r' in E do
7:       if IsMatched(currentRecord,r') then
8:          buddy = r'
9:          break
10:   if buddy == null then
11:      E.append(currentRecord)
12:   else
13:      merged = Merge(currentRecord, buddy)
14:      E.remove(buddy)
15:      R.append(merged)
16: return E
```

4.2 Linkage: F-Swoosh

Swoosh is an entity resolution approach which compares records according to features (a feature is a set of attributes) selected by the user (Benjelloun et al. 2009). A pair of records are merged into a new record when one of their features meets the matching criteria provided by the user. The two original records are removed after a new record is created by merging. Algorithm 2 is a basic description of the algorithm. As a version of Swoosh algorithm, F-Swoosh is optimised with various hashtables and feature operations. In practice we use F-Swoosh for matching, but here we use the algorithm of R-Swoosh to present this approach as R-Swoosh is simpler and more straightforward. The function $IsMatched(r, r')$ in line 7 is a function provided by the user which compares two records and decides if they are a match. The $Merge(r, r')$ function in line 13 is provided by the user as well, which decides how to merge each attribute of two records r and r' (possible ways to handle attribute merging include keeping the latest value, or creating a list of all values (Benjelloun et al. 2009)).

4.2.1 Tracking Merged Records

While F-Swoosh merges records that likely belong to the same entity, we also keep track of the time (which can be a year, such as 2012, or a date, such as 20-10-2011) when each attribute value was originally generated. Each attribute value is stored in the form of a tuple: (*attribute_value*, *updated_date*), and each attribute can have one-to-many attribute values for a given record after multiple merges.

Example: Let record r_1 = ['Tom', 'Bruce', '21 May Street', '2012-12'] (in the format of [*first name*, *last name*, *address*, *time-stamp*]), record r_2 = ['Tom', 'Steven', '21 May Street', '2013-06']. The new record created by merging r_1 and r_2 will be: [[('Tom','2012-12'), ('Tom','2013-06')], [('Bruce','2012-12'), ('Steven','2013-06')], [('21 May Street','2012-12'), ('21 May Street','2013-06')]].

4.3 Temporal Models

Temporal models refer to the models that adjust record pair similarity using temporal information (such as time related rules and patterns) learned from a dataset. In this paper, all temporal models used are built from a training dataset.

The *decay model* calculates disagreement decay and agreement decay of each attribute, and uses them to calculate the similarity of a record pair (Li et al. 2012). Disagreement decay and agreement decay both describe attribute characteristics learned from training data, indicating the probability for an attribute to change within a time distance, and the probability for an attribute to be shared by multiple entities within a time distance, respectively. Time distance refers to the difference between two time-stamps. Time distance is measured by a time unit which is defined by the user, such as days, years, or hours.

A *life span* l refers to the time distance between the time-stamps of two values. An attribute value's life span is *full* when the value has a date when it is start to be used, and another date when it is changed to another value. The time distance between the first date and the second date is a *full life span*, denoted as l_f. Similarly, if the attribute's value does not change between two time-stamps, the time distance between the two time-stamps is a *partial life span*, denoted l_p.

For example, assume an entity with three different last names over five records, with time-stamps in the form of [year-month]: 'Taylor' (2011-10) → 'Taylor' (2011-12) → 'Spire' (2012-12) → 'Spire' (2013-10) → 'Wright' (2015-10). The time distance between the first and the third record is one full life span with a length of 14 months (2011-10 to 2012-12), and the distance between the third and the fifth record is another full life span with a length of 34 months (2012-12 to 2015-10). Note in this example, month is being used as a time unit but it is not necessary for all datasets. From the example above, the time distance between the first and the second record is a partial life span with a length of 2 months (the time distance between 2011-10 and 2011-12), and the time distance between the third record and the forth record is another partial life span with a length of 10 months.

$\bar{L}_f$ denotes the list of all full life spans of an attribute A for all entities. Similarly, $\bar{L}_p$ denotes the list of all partial life spans of an attribute A for all entities.

Definition 4.1. (Disagreement Decay $d^{\neq}$): Let Δt be a time distance, $A \in \mathbf{A}$ be a single valued attribute. The disagreement decay of A over time Δt, is the probability that an entity changes its value of A within a time distance Δt (Li et al. 2012).

$$d^{\neq}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|} \tag{1}$$

Equation 1 calculates the disagreement decay $d^{\neq}$ given an attribute A and a time distance Δt. $\bar{L}_f$ is the set of full life spans of attributes values of attribute A. The time unit of Δt, is defined by the user, which can be days, months, or years.

Definition 4.2. (Agreement Decay $d^{=}$): Let Δt be a time distance, $A \in \mathbf{A}$ be a single valued attribute. The agreement decay of A over a time distance Δt, is the probability that two different entities share the same value for A within Δt. (Li et al. 2012)

$$d^{=}(A, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|} \tag{2}$$

Equation 2 calculates the agreement decay $d^{=}$. $\bar{L}$ is a list of life spans. For each record from a training dataset, if it has the same attribute value with another record which belongs to a different entity, the time distance between the two records is added to $\bar{L}$. If no entity has the same attribute value, a life span with length ∞ is added to $\bar{L}$.

We use agreement decay and disagreement decay to calculate w_A (weight per attribute), as shown in Equation 3. Then, weights are used to calculate the pair-wise similarity between two records. As shown in Equation 4, $s_d(r, r')$ denotes the decay adjusted similarity between two records r and r'. s_a refers to the similarity between a pair of attribute values.

$$w_A(s_a, \Delta t) = 1 - s_a \cdot d^{=}(A, \Delta t) - (1 - s_a) \cdot d^{\neq}(A, \Delta t) \tag{3}$$

$$s_d(r, r') = \frac{\sum_{A \in \mathbf{A}} w_A(s_a(r.A, r'.A), |r.t - r'.t|) \cdot s_a(r.A, r'.A)}{\sum_{A \in \mathbf{A}} w_A(s_a(r.A, r'.A), |r.t - r'.t|)} \tag{4}$$

5 Our Approach

This section discusses our temporal model and training strategy in detail.

5.1 Disagreement Probability

Disagreement probability is a concept introduced in this work. It has a similar definition as disagreement decay (as shown in Equation 1), but is modified to make it easier to be used with a machine learning model. From Equation 5, we can see that the only difference between $d^{\neq}_{prob}$ and $d^{\neq}$ is that the divisor no longer decreases with an increasing Δt. We use $d^{\neq}_{prob}$ instead of $d^{\neq}$ because the equation of $d^{\neq}_{prob}$ has a divisor that is fixed for each entity. This problem is therefore intuitively easier to fit into a machine learning classifier that when a life span l is encountered, we can immediately decide if it is lower than a Δt and create a training instance with the attribute values associated to the life span l.

$$d^{\neq}_{prob}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\bar{L}_p|} \tag{5}$$

$d^{\neq}_{prob}$ is normalized into the range $[0, 1]$, and then used as weights to adjust respective attribute-wise similarities, as shown in Equation 6. s_p denotes the adjusted similarity between a pair of records, and the function $s_a(a, a')$ returns the similarity between a pair of attribute values. The specific value comparison function for an attribute is defined by the user, which returns a similarity measure in the range [0,1]. These comparison functions can be approximate string similarity functions, such as edit-distance, Jaro-Winkler, etc. (Christen 2012*a*).

$$s_p(r, r') = \sum_{A \in \mathbf{A}} \frac{1 - d^{\neq}_{prob}(A, |r.t - r'.t|)}{\sum_{A' \in \mathbf{A}} 1 - d^{\neq}_{prob}(A', |r.t - r'.t|)} \cdot s_a(r.a, r'.a) \tag{6}$$

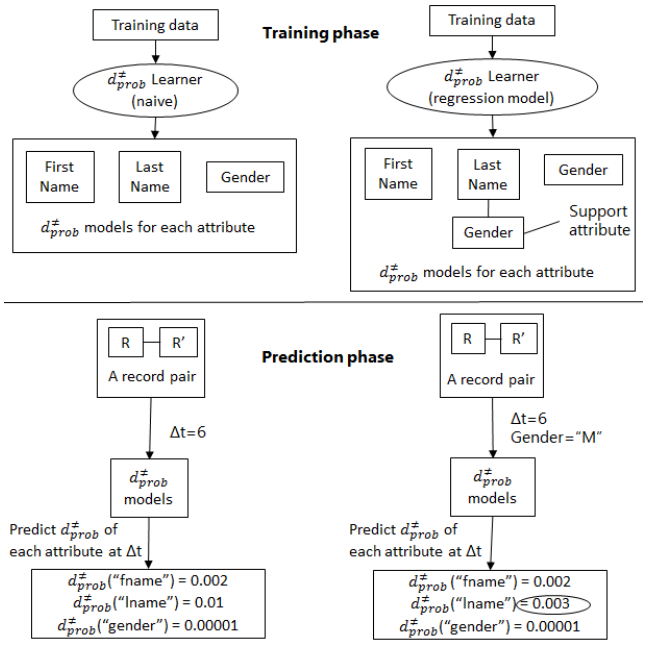


Figure 3: Disagreement probability training and prediction with decay model (left) and regression model (right). We can see the model on the right takes a value from one support attributes as input, *gender* in this case, and calculates a disagreement probability value for a specific subset of entities (males in this example). Whereas the model on the left side calculates a disagreement probability value regardless to the entity's gender. The number in the black circle shows that a different disagreement probability value is calculated for attribute *last name*.

5.2 A Regression Model

From the previous equations we can see that agreement decay, disagreement decay, and disagreement probability are calculated with only one attribute independently each time. For example, when the disagreement decay of attribute *last name* is calculated, the temporal model calculates the overall probability for an entity to change its last name within a given interval Δt. However, the probability of an entity to change its last name is often associated with gender and age. The disagreement decay for last name, calculated without considering its gender and age group, will for example be too high for older male entities, and too low for younger female entities, because it is rare for an older male person to change last name, but common for a young female to change last name due to marriage.

Support attributes are assigned to certain attributes when the value of a support attribute may affect the probability of the attributes to change. For example, when building a model which predicts the probability for attribute *address* to change, attribute(s) such as *gender* and *age* can be used to make the prediction more accurate. Support attributes are selected by the user based on empirical knowledge, however they can also be selected by a feature selection strategy (Blum & Langley 1997).

Figure 3 compares the difference between the original learning strategy (left) and the learning strategy using a machine learning model (right), with attribute *gender* used as a support attribute for attribute *last name*.

Algorithm 3 shows how we use a machine learning model to predict disagreement probability. In line 6, the function $getAttValues(r, L_A)$ extracts a list of attribute values from a record r according to an attribute list L_A. The attribute values are important features later used to create training instances. In lines 9-12, a training instance with class value 0, with Δt and $supportValues$ as features is created. Since partial life spans indicate no value change, we use 0 to denote *non-change*. In lines 16-17, in the case that a change happen within a Δt, we create a training instance with class value 1 which denotes *change*. The training instances are then sent to the machine learning model defined by the user which can be used to predict $d^{\neq}_{prob}$ for an attribute A.

Figure 4 shows the different decay values calculated for attribute *last name* using disagreement decay $d^{\neq}$ (see Equation 1) and disagreement probability $d^{\neq}_{prob}$ predicted by a regression model which was trained using Algorithm 3. Attribute *gender* was used as the support attribute. We can see that with our approach, lower disagreement probabilities are calculated for entities with male gender, whereas higher disagreement probabilities are calculated for entities with female gender, for attribute *last name*. This makes sense as female entities change last name more often than male entities due to marriages.

Algorithm 3 Temporal model training with disagreement probability

Input:
- A list of clusters: **C**, each cluster $C \in \mathbf{C}$ contains a list of records that belong to an entity
- An attribute to build temporal model: A_1
- A list of support attributes: L_A
- A machine learning algorithm: *model*

Output:
- A trained temporal model

```
1:  train = ∅
2:  for C in C do
3:      start = 1
4:      while start ≤ |C| do
5:          end = start +1
6:          supportValues = getAttValues(C[start], L_A)
7:          while C[start].a_1 == C[end].a_1 and end ≤ |C| do
8:              end = end + 1
9:          if end > |C| then
10:             Δt = C[|C|].t − C[start].t + 1 // Partial life span
11:             for i = 1 to Δt_max do
12:                 train.add([0, Δt, supportValues])
13:         else
14:             Δt = C[end].t − C[start].t // Full life span
15:             for i = 1 to max_Δt do
16:                 if Δt ≥ i then
17:                     train.add([1, Δt, supportValues])
18:                 else
19:                     train.add([0, Δt, supportValues])
20:         start = end
21: // Train the model with the accumulated training data
22: model.fit(train)
23: return model
```

5.3 Combine Disagreement Probability with Agreement Decay

Disagreement probability can be modified into the same form as disagreement decay by normalising it: $d^{\neq}_{nprob}(A, \Delta t) = \frac{d^{\neq}_{prob}(A,\Delta t)}{max(d^{\neq}_{prob}(A))}$. where $max(d^{\neq}_{prob}(A))$ is the maximum disagreement probability over all Δt. Using Equation 3 and Equation 4 above, the temporal similarity for our model can be calculated by substituting $d^{\neq}$ with $d^{\neq}_{nprob}$.

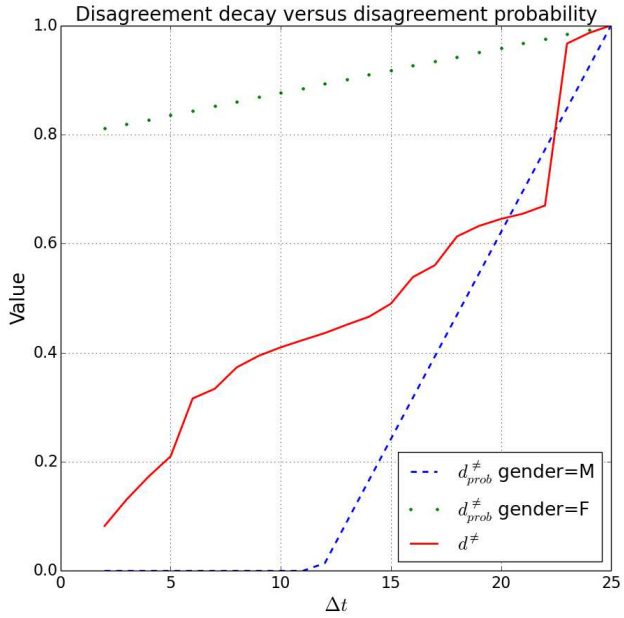


Figure 4: Different decay values for attribute *last name* calculated using disagreement decay $d^{\neq}$ and disagreement probability $d^{\neq}_{prob}$ predicted by a regression model. Attribute *gender* was used as the support attribute.

Table 1: A summary of snapshots of the NCVR database and temporal records created from each snapshot

Snapshot (year-month)	Number of records	Records added	Records updated	Temporal records
2011-10	6,233,683.00	5,660,833	0	5,660,833
2011-12	6,981,774.00	285,091	50,181	335,272
2012-02	6,974,887.00	40,370	40,187	80,557
2012-04	7,054,734.00	82,370	82,496	164,866
2012-06	7,090,377.00	38,792	87,326	126,118
2012-08	7,134,332.00	47,928	86,418	134,346
2012-10	7,310,212.00	183,858	182,609	366,467
2012-12	7,524,471.00	221,922	354,053	575,975
2013-02	7,251,818.00	34,607	83,552	118,159
2013-04	7,268,064.00	28,878	43,423	72,301
2013-06	7,291,726.00	27,293	26,511	53,804
2013-08	7,325,036.00	35,382	32,528	67,910
2013-10	7,358,266.00	35,683	54,264	89,947
2013-12	7,388,104.00	33,495	50,440	83,935
2014-02	7,391,221.00	28,336	32,974	61,310
2014-06	7,453,901.00	69,002	120,422	189,424
2014-08	7,490,428.00	38,250	50,923	89,173
2014-10	7,539,857.00	53,438	86,702	140,140
2014-12	7,608,324.00	72,576	211,607	284,183
2015-02	7,111,324.00	29,247	43,555	72,802
2015-04	7,129,997.00	24,578	50,048	74,626
2015-06	7,156,299.00	31,320	88,298	119,618
2015-08	7,198,755.00	45,845	58,748	104,593
2015-10	7,244,629.00	50,579	71,549	122,128
2015-12	7,272,428.00	46,563	74,142	120,705
2016-02	7,313,555.00	45,498	77,645	123,143

6 Experiments

In this section we describe datasets, methods and measures used in our experiment. Then we present and discuss the experimental results.

6.1 Experimental Settings

we describe the characteristics of the datasets we used in the experiments and the methods, measures, and other implementation details of the experiments.

Table 3: Testing datasets

Dataset name	Number of entities	Number of records	Entities with more than one records
Avery	13,707	15,464	1,604
Buncombe	221,106	282,947	49,490
Cherokee	25,450	28,715	2,875
Gates	9,396	10,504	1,006
Guilford	410,661	515,878	85,192
Montgomry	18,686	21,544	2,539
Avery_L2	1,604	3,361	1,604
Buncombe_L2	49,490	111,331	49,490
Cherokee_L2	2,875	6,140	2,875
Gates_L2	1,006	2,114	1,006
Guilford_L2	85,192	190,409	85,192
Montgomry_L2	2,539	5,397	2,539

Temporal datasets: The temporal datasets we used in this paper are from the North Carolina Voter Registration (NCVR) datasets collected every two months[1]. The datasets have ground truth (entity identifiers) available for all entities. Because the raw datasets are collected in the form of snapshots of databases, we preprocessed them to refine their temporal aspects. Only records that describe changes of an entity are selected into the temporal dataset.

If an entity never has changes in its attribute values, only the earliest record will be selected for that entity. If we sort records of an entity by increasing time-stamp values, the first record of an entity is an *add*, since it indicates the time that the entity was created in that dataset. Any following record of that entity, where a record has any attribute values (except age and time-stamp) that are different from the last record, then this record is considered as *update*.

Table 1 summarizes the number of records in each database snapshot and the number of temporal records created from them. A total of 8,336,205 unique entities were added to the datasets at different point of time.

Example: As shown in Table 2, an entity $e1$ with name "William Crawford Taylor" was added to the snapshot at 2011-10, but only two updates occurred over the next four snapshots: middle name was changed from 'Rose' to 'Louise' at 2012-02, and last name was changed from 'Taylor' to 'Clark' at 2012-06.

The temporal dataset of entity $e1$ will only have the initial record and the two records when the updates occurred, as shown in the left side of the table. The second entity, $e2$, used another first name at 2011-12 and 2012-02, then was changed back. This entity will have three temporal records, one for the first record of the entity at 2011-10, two for the two updates. For the third entity $e3$ whose name "Michelle Mary Lee" has never been changed across the five snapshots, this entity will only have one record in the temporal dataset as there was no update happened.

Above 80% of entities are in the same situation as the third entity $e3$ in the NCVR datasets, indicating updating election information is not very common for most people.

Only a subset of datasets and attributes were used in our work. These attributes were used in the test: last name, middle name, first name, name suffix, residential address, age, and sex code. Sex code has been used as a support attribute for last name, middle name, and residential address. We choose these attributes as they are commonly seen in different datasets, comparing to other attributes, such as race code, party code, and phone number. We did not use

[1] http://dl.ncsbe.gov/

Table 2: Sample records in a temporal dataset

	Raw records				Temporal records				
Entity	**Date**	**First name**	**Last name**	**Middle name**	**Date**	**First name**	**Last name**	**Middle name**	**Action**
e1	2011-10	William	Taylor	Rose	2011-10	William	Taylor	Rose	Add
e1	2011-12	William	Taylor	Rose					
e1	2012-02	William	Taylor	Louise	2012-02	William	Taylor	Louise	Update
e1	2012-04	William	Taylor	Louise					
e1	2012-06	William	Clark	Louise	2012-06	William	Clark	Louise	Update
.....									
e2	2011-10	David	Edward	Jr	2011-10	David	Edward	Jr	Add
e2	2011-12	Dave	Edward	Jr	2011-12	Dave	Edward	Jr	Update
e2	2012-02	Dave	Edward	Jr					
e2	2012-04	David	Edward	Jr	2012-04	David	Edward	Jr	Update
e2	2016-06	David	Edward	Jr					
.....									
e3	2011-12	Michelle	Lee	Mary	2011-12	Michelle	Lee	Mary	Add
e3	2012-02	Michelle	Lee	Mary					
e3	2012-04	Michelle	Lee	Mary					
e3	2012-06	Michelle	Lee	Mary					
.....									

the full datasets for testing at this stage as the proposed approaches are still being tuned and testing on the full datasets is time consuming. The results from selected subsets are well informative so far. We aim to test on the full datasets in the future.

Training dataset. The NCVR temporal dataset of county 'Alexander' was used as a training dataset. Which has 33,995 records from 27,725 entities, and 5,403 entities have at least two records.

Testing datasets: Temporal datasets of six counties were selected from NCVR as testing datasets, where each county has two versions of temporal datasets: the original temporal dataset (named by the county's name) and a refined temporal dataset (L2 dataset) where every entity has at least two records, as shown in Table 3. The original temporal datasets of NCVR have a low percentage of entities who have at least two records, which means the majority of records are not linkable. L2 versions of datasets were created by extracting records from entities with at least two records from its respective original temporal dataset, to test the algorithm's performance when all of the records are linkable.

The reason to create one L2 dataset for each county is to test the algorithm's performance in a distinct data environment where most entities have multiple records.

6.1.1 Implementation

We implemented all algorithms in Python 2.7, and the experiments were conducted on a server with 64-bit Intel Xeon (2.4 GHz) CPUs, 128 GBytes of memory and running Ubuntu 14.04.

We implemented four algorithms which are being discussed below. All of the algorithms above were implemented on the R-Swoosh clustering framework (Benjelloun et al. 2009). Blocks were generated using LSH as discussed in Section 4.1, with pairs completeness greater than 99%, which means greater than 99% of records can be correctly linked with an ideal linkage technique. The same set of blocks was used by the four algorithms. For the regression model, we used linear regression model from sklearn python package with default settings. [2]

For string attributes, the similarity of a pair of attribute values was calculated using the Jaro-Winkler string comparison function (Christen 2012*a*). The similarity of a pair of age values was calculated as: $s_{age} = \frac{1}{|age_1 - age_2|+1}$. The similarity threshold used by all algorithms was 0.8, which means record pairs with similarity equal to or above this threshold are matches (same entity) and below are non-matches. This threshold is arbitrarily chosen. Future experiments can be done with different similarity thresholds.

- No model. A baseline approach with no temporal model. Weights of attributes were not adjusted by a temporal model.
- Decay model (*Decay*). A baseline approach using the temporal model proposed by Li et al. (2012). The algorithm calculates a disagreement rate (the probability for an attribute to change within a certain time interval) and agreement rate (the probability for an entity's attribute value to be the same as other different entities within a time interval), and uses the two rates to adjust the weight of each attribute.
- Disagreement probability regression model (*Disprob*). A temporal model uses a regression model to predict disagreement probability, and reduces the weight of attributes when its predicted disagreement probability is high. The weights of attributes are normalized so that the sum of attribute weights is always 1.
- Disagreement probability plus agreement decay regression model (*Mixed*). With a disagreement probability being predicted in the same way as the method above, the mixed method also calculates agreement decay from the decay model. Disagreement probability and agreement decay are combined to adjust the weight of each attribute.

Measures. Let *res* be a linkage result in the form of clusters of records that are matching, *stand* be the ground truth that *res* corresponds to, which is also in the form of clusters of records. Pair-wise precision $(Precision) = \frac{|res \cap stand|}{|res|}$, pair-wise recall $(Recall) = \frac{|res \cap stand|}{|stand|}$, and $F_1 = \frac{2 * Precision * Recall}{Precision + Recall}$.

6.2 Experimental Results

We compared the four approaches (two baseline and two proposed approaches) on the 12 testing datasets, using precision, recall, and F1. The 'Alexander' dataset was used as the training dataset.

[2] http://scikit-learn.org

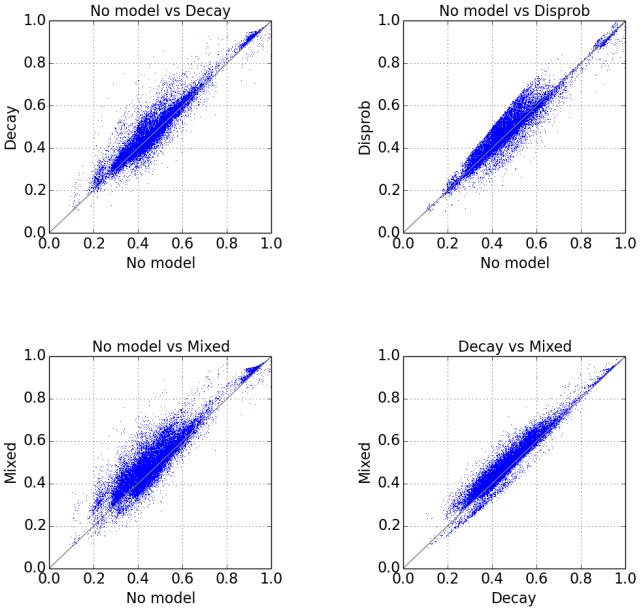
Table 4: Linkage results on original temporal datasets

Dataset	Avery			Buncombe			Cherokee		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
No model	**0.8830**	0.9350	**0.9083**	**0.7495**	0.9517	**0.8386**	**0.8607**	0.9198	**0.8893**
Decay	0.6857	0.9579	0.7992	0.5740	0.9604	0.7185	0.6177	0.9394	0.7453
Disprob	0.8449	0.9126	0.8774	0.7407	0.9315	0.8252	0.8000	0.8896	0.8425
Mixed	0.6421	**0.9802**	0.7759	0.5158	**0.9754**	0.6747	0.5687	**0.9658**	0.7159
Dataset	Gates			Guilford			Montgomry		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
No model	**0.8840**	0.9326	**0.9076**	**0.7100**	0.9421	**0.8097**	**0.8904**	0.9272	**0.9085**
Decay	0.7101	0.9581	0.8157	0.5741	0.9531	0.7166	0.7269	0.9450	0.8217
Disprob	0.8346	0.8998	0.8660	0.7072	0.9219	0.8004	0.8492	0.9002	0.8740
Mixed	0.6654	**0.9737**	0.7905	0.5037	**0.9707**	0.6632	0.6682	**0.9698**	0.7912

Table 5: Linkage results on L2 temporal datasets

Dataset	Avery_L2			Buncombe_L2			Cherokee_L2		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
No model	0.9841	0.9339	0.9584	0.8922	0.9517	0.9210	**0.9907**	0.9195	0.9538
Decay	0.9766	0.9568	0.9666	0.8491	0.9604	0.9013	0.9730	0.9421	0.9573
Disprob	**0.9882**	0.9126	0.9489	**0.9297**	0.9315	**0.9306**	0.9863	0.8894	0.9353
Mixed	0.9597	**0.9787**	**0.9691**	0.7875	**0.9757**	0.8716	0.9564	**0.9680**	**0.9621**
Dataset	Gates_L2			Guilford_L2			Montgomry_L2		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
No model	**0.9887**	0.9334	0.9603	0.8538	0.9428	0.8961	**0.9832**	0.9269	0.9542
Decay	0.9701	0.9589	0.9645	0.8319	0.9540	0.8888	0.9657	0.9456	0.9555
Disprob	0.9856	0.8998	0.9407	**0.9074**	0.9225	**0.9149**	0.9814	0.9005	0.9392
Mixed	0.9619	**0.9737**	**0.9677**	0.7597	**0.9716**	0.8527	0.9457	**0.9701**	**0.9578**

Figure 5: Similarity scatter plot between the four approaches. Each plot shows the similarities produced by the two approaches for each of the record pairs. We can see that *decay* and *mixed* tend to produce a higher similarity than *no model*, and *mixed* tends to produce a higher similarity than *decay*.



Results based on original datasets: Table 4 shows the results on the six original temporal datasets (shown in Table 3). We observed that the performance of a linkage approach drops significantly when the size of a dataset increases. This is expected, as the larger a dataset is, the more likely for a non-matched pair with high similarity exists.

Results based on L2 datasets: Table 5 shows the results on the six L2 datasets (where only entities with at least two records are included). L2 datasets performed significantly better than original datasets since there is at least one match guaranteed for each record and they are also smaller.

We observed that the approach without a temporal model (*no model*) produced the best F1 scores, which is consistent with the observation from Li et al. (2012) where better performance was achieved only after using a temporal clustering technique. The *mixed* approach produced the highest Recall throughout the experiment, however at a significant cost with Precision. The *disprob* approach produced the best result among temporal models, however, generally inferior to the *no model* baseline.

Comparison between temporal and non-temporal approaches. We extracted a subset of the results in attempt to analyze the impact of the temporal models. As Figure 5 shows, we found the approach with a temporal model tends to produce higher similarity on record pairs than the approach without a temporal model. On a closer look we found that temporal models gave attribute *first name* a higher weight and attribute *address* and *last name* a lower weight, which is expected since it is more common for people to change address and last name. However, when a pair of non-match records share a popular first name such as 'Anna', the temporal model made them more likely to be matched by mistake, especially when their age and gender are the same too. A potential fix to this issue is to introduce a frequency based weighting strategy, such as weighting attribute according to a value's frequency in the context (TF-IDF) (Witten et al. 1999), or use the temporal clustering method as proposed by Li et al. (2012).

7 Conclusion and Future Works

In this paper we developed two attribute weighting approaches using a linear regression model to improve the quality of temporal record linkage. Our regression model uses multiple attribute values from a record pair as input, to predict the probability of an attribute value to change within a certain time period, and then adjust the weight of the attribute accordingly. We evaluated our approaches on twelve datasets derived from NCVR datasets. Experimental results show that one of our approaches performed better than the temporal baseline, and another approach achieved overall highest recall.

In the future, we intend to incorporate a frequency based weighting strategy into the framework, and to see if the undesired high similarities can be adjusted properly. Another possible direction is to test the temporal models with different clustering techniques, such as the temporal clustering techniques proposed by Li et al. (2012) and Chiang et al. (2014*b*).

Acknowledgements

This work was partially funded by the Australian Research Council (ARC) under Discovery Project DP160101934.

References

Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S. E. & Widom, J. (2009), 'Swoosh: a generic approach to entity resolution', *The VLDB Journal* **18**(1), 255–276.

Blum, A. L. & Langley, P. (1997), 'Selection of relevant features and examples in machine learning', *Artificial Intelligence* **97**(1-2), 245–271.

Chiang, Y.-H., Doan, A. & Naughton, J. F. (2014*a*), Modeling Entity Evolution for Temporal Record Matching, *in* 'ACM SIGMOD', New York, NY, USA, pp. 1175–1186.

Chiang, Y.-H., Doan, A. & Naughton, J. F. (2014*b*), 'Tracking Entities in the Dynamic World: A Fast Algorithm for Matching Temporal Records', *Proceedings of the VLDB Endowment* **7**(6), 469–480.

Christen, P. (2012*a*), *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Springer, Berlin Heidelberg.

Christen, P. (2012*b*), 'A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication', *IEEE Transactions on Knowledge and Data Engineering* **24**(9), 1537–1555.

Christen, P. & Gayler, R. W. (2013), Adaptive Temporal Entity Resolution on Dynamic Databases, *in* 'PAKDD, Springer LNAI', Gold Coast, Australia, pp. 558–569.

de Vries, T., Ke, H., Chawla, S. & Christen, P. (2011), 'Robust Record Linkage Blocking Using Suffix Arrays and Bloom Filters', *ACM TKDD* **5**(2), 1–27.

Indyk, P. & Motwani, R. (1998), Approximate nearest neighbors: towards removing the curse of dimensionality, *in* 'ACM Symposium on Theory of Computing', Dallas, TX, USA, pp. 604–613.

Kim, H.-S. & Lee, D. (2010), HARRA: Fast Iterative Hashed Record Linkage for Large-scale Data Collections, *in* 'International Conference on Extending Database Technology', ACM, Lausanne, Switzerland, pp. 525–536.

Krueger, D., Montgomery, D. C., Peck, E. A. & Vining, G. G. (2015), *Introduction to Linear Regression Analysis*, John Wiley & Sons., Hoboken, NJ.

Kum, H.-C., Krishnamurthy, A., Machanavajjhala, A. & Ahalt, S. C. (2014), 'Social Genome: Putting Big Data to Work for Population Informatics', *IEEE Computer* **47**(1), 56–63.

Li, F., Lee, M. L., Hsu, W. & Tan, W.-C. (2015), Linking Temporal Records for Profiling Entities, *in* 'ACM SIGMOD', New York, NY, USA, pp. 593–605.

Li, P., Dong, X. L., Maurino, A. & Srivastava, D. (2012), 'Linking temporal records', *Frontiers of Computer Science* .

Rajaraman, A. & Ullman, J. D. (2011), *Mining of Massive Datasets*, Cambridge University Press, Cambridge.

Ukkonen, E. (1992), 'Approximate string-matching with q-grams and maximal matches', *Theoretical Computer Science* **92**(1), 191–211.

Wang, Q., Cui, M. & Liang, H. (2016), 'Semantic-Aware Blocking for Entity Resolution', *IEEE Transactions on Knowledge and Data Engineering* **28**(1), 166–180.

Witten, I. H., Moffat, A. & Bell, T. C. (1999), *Managing Gigabytes: compressing and indexing documents and images*, Morgan Kaufmann Publishers, San Francisco, Calif.