*Short Paper*

# Repairing of Record Linkage: Turning Errors into Insight*

Quyen Bui-Nguyen, Qing Wang, Jingyu Shao, Dinusha Vatsalan
Australian National University, Canberra, Australia
quyen.m.bui.nguyen@gmail.com,{qing.wang,jingyu.shao,dinusha.vatsalan}@anu.edu.au

## ABSTRACT

Linking records from different data sources, referred to as record linkage, is a longstanding but not yet satisfactorily resolved question in many fields of science. For practitioners, it is difficult to ensure the quality of linkage at the time of applying linkage techniques in real world applications. Instead, linkage errors are often detected later on, mostly by users of the applications. This not only requires us to repair errors, but also provides us with opportunities to observe the linkage quality and uncover why such errors occur. In viewing that record linkage is a complex and evolving process, we study how to acquire insights from linkage errors for achieving high-quality linkage. We propose a generic repairing framework which allows us to start with imperfect linkage models, and dynamically repair linkage models and errors for improved linkage quality. We have evaluated our repairing framework over three real-world datasets and the experimental results show that the performance of the proposed tree-structured classifier SVM-tree outperforms the baseline methods.

## 1 INTRODUCTION

Determining which records from one or more datasets correspond to the same real-world entities, referred to as *record linkage*, is a fundamental problem arising in many fields of computer science, e-commerce, health and social sciences [4]. Current research on record linkage mainly focuses on developing accurate and efficient linkage techniques, which are constrained by a number of factors (e.g., concerns about quality of data, ambiguity of domain knowledge and unavailability of training labels) and fails to capture the full variety of record linkage [8, 11]. Therefore, it is generally difficult to guarantee linkage quality at the time when record linkage techniques are applied.

One question that often arises is how to handle linkage errors which cannot be detected at the time of applying linkage techniques, but are reported later on, particularly by users of the record linkage based applications. Since data may be enriched with additional information over time, we can find linkage errors that are hidden in the linkage process. Essentially, record linkage is not a static and one-off task, but rather a complex, continuous and evolving process. Based on the insight acquired from linkage errors we may build a repairing framework to improve linkage quality through repairing errors.

Nevertheless, building such a repairing framework is challenging. Can we generalise errors into insight and leverage such insight into improving linkage models? Not all errors are equally useful for finding insight. Some errors may be outliers and generalising such errors may even deteriorate the performance of linkage models. Thus, we need a means for assessing the genericity of linkage errors - to what extent a linkage error can act as being a representative example of similar linkage errors. In this paper, we propose a dynamic repairing framework, which aims to turn errors into insight for improved linkage models and results. The central idea is to leverage detected errors to derive similar and relevant errors for maximally repairing an "imperfect" linkage model. Figure 1 presents a high-level overview of our repairing framework which incorporates a tree-structured classifier and an error analyser for supporting an iterative repairing process. More specifically, linkage errors such as false matches or non-matches may be detected by a user or governed by integrity rules and sent to an error analyser. The error analyser uses an error normality measure to determine whether an error can be generalised for improving the linkage model. Then qualified linkage errors are sent to the tree-structured classifier, and through refining the tree structure of the classifier, the linkage quality can be improved.

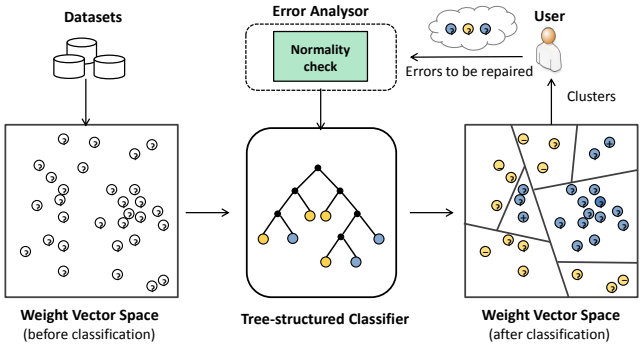


**Figure 1: Proposed repairing framework, where each circled question mark represents an unlabeled weight vector, and each weight vector labeled as matches is shown with a blue ⊕, and non-matches with a yellow ⊖.**

Our contributions are as follows: (1) We develop a generic repairing framework which allows us to start with imperfect linkage models and actively repair linkage models and errors for improved linkage quality. (2) We design a novel measure to assess the genericity of an linkage error, which indicates the probability of generalising errors into insight. (3) We have experimentally validated the effectiveness of the proposed repairing framework using several real-world datasets.

## 2 RELATED WORK

Record Linkage (RL) has long been central to the study of data integration and data cleaning [2, 5, 13]. Previous research has studied various aspects of the RL process such as: blocking, similarity comparison, classification, evaluation, active learning, and training data selection [6, 7, 9, 10, 12]. Nonetheless, these works primarily focused on preventing rather than repairing errors in linkage results.

In this paper, instead of only repairing detected errors, we also study the problem of repairing linkage models through detected linkage errors, i.e., linkage errors are leveraged to improve the performance of linkage models. It is worthy to note that,

*Produces the permission block, and copyright information

different from database repair [1, 3], we do not stipulate a minimal change requirement on our repairs. Existing approaches of database repair mostly deal with data errors either by obtaining consistent answers to queries [1] or by developing automated methods to find a repair that satisfies required constraints and also "minimally" differs from the original database [3]. These approaches are often computationally expensive and not applicable to repairing linkage models in real-world applications.

## 3 PROBLEM STATEMENT

Let $R$ be a finite, non-empty set of records from one or more data sets. We assume that a set $F = \{f_1, \ldots, f_n\}$ of features is selected for performing record linkage tasks and each record $r \in R$ has a number of feature values $(v_1, \ldots, v_n)$ over $F$. Accordingly, each pair of records $(r_i, r_j)$ over $R$ is associated with a n-dimensional *weight vector* $\mathbf{x_{ij}} = (x_1, \ldots, x_n) \in [0, 1]^n$ where each $x_k$ represents the similarity between the feature values of $f_k$ in $r_i$ and $r_j$. For example, a pair of records $(r_1, r_2)$ may have three features $\{fname, sname, age\}$ with $r_1.fname = Rob$, $r_1.sname = Smith$, $r_1.age = 30$, $r_2.fname = Robert$, $r_2.sname = Smith$ and $r_2.age = 31$, and correspond to a 3-dimensional weight vector $(0.5, 1, 0.5)$.

Let $X$ consist of the weight vectors to which all pairs of records in $R$. Some blocking technique may also be applied for efficiency [9, 11]. In viewing that weight vectors are a set of data points in $X$, a *weight vector space* $(X, \delta)$ can be defined for $X$ together with a *similarity metric* $\delta : X \times X \mapsto [0, 1]$ that satisfies the non-negativity, identity, symmetry and triangle inequality properties as defined in [14]. Thus, the higher two data points $\mathbf{x_1}$ and $\mathbf{x_2}$ are similar, the larger the value $\delta(\mathbf{x_1}, \mathbf{x_2})$ is. A *partition* of $X$ is a set $\{X_1, \ldots, X_q\}$ of pairwise disjoint subsets with $\bigcup_{1 \le i \le m} X_i = X$. We call each $(X_i, \delta)$ $(i \in [1, q])$ a *weight vector subspace* of $(X, \delta)$.

The label of a weight vector $\mathbf{x} \in X$ is denoted as $y(\mathbf{x})$ and $y(\mathbf{x}) \in \{1, -1\}$, where 1 refers to a true match and $-1$ refers to a true non-match. In this paper, we consider a linkage model as a binary classifier $c : X \rightarrow \{-1, 1\}$, which can classify a weight vector $\mathbf{x}$ either as 1 or as -1. Thus, two kinds of errors may occur in record linkage: (1) false positives (i.e. false matches) and (2) false negatives (i.e. false non-matches):

- $fp(c) = \{\mathbf{x} \in X | c(\mathbf{x}) = -1 \wedge y(\mathbf{x}) = 1\}$;
- $fn(c) = \{\mathbf{x} \in X | c(\mathbf{x}) = 1 \wedge y(\mathbf{x}) = -1\}$.

*Definition 3.1.* (RL repair problem) Let $c$ be a linkage model and $E$ be a set of detected errors. Then the *RL repair problem* is to find a linkage model $c^*$ such that the following objective function is minimised:

$$\operatorname*{argmin}_{c} \frac{fp(c^*) + fn(c^*)}{fp(c) + fn(c)} \tag{1}$$

The focus of the RL repair problem is on maximally repairing linkage models through detected errors, i.e., gain maximum insights from errors, rather than repairing only detected errors.

## 4 REPAIRING FRAMEWORK

We propose a novel framework for solving the RL repair problem.

### 4.1 Error Analyser

Generally, linkage errors may be caused by various reasons, e.g., poor data quality, biased classifiers and insufficient training data. Although it is desirable to repair all errors, linkage errors are not equally informative for improving a linkage model. For instance, repairing a linkage model based on errors that are indeed outliers often leads to the overfitting problem. On the other hand,

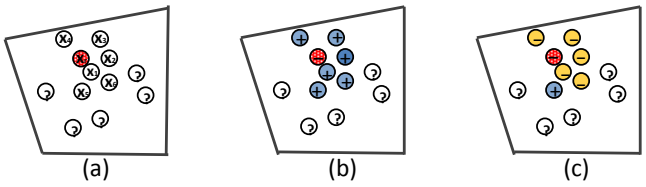


**Figure 2: A false match, denoted as $\mathbf{x_r}$ and highlighted as being red, is detected in a weight vector subspace $X_i$.**

errors that repeatedly occur under certain similar conditions may suggest a pattern of these errors which is more informative for repairing a linkage model. To quantify the informativeness of errors, we thus develop the notion of error normality to measure the degree errors can be generalised from specific cases to more structured ones.

*Definition 4.1.* Let $(X, \delta)$ be a weight vector subspace, $\mathbf{x} \in X$, and $N(\mathbf{x}, X)$ refer to the set of k nearest neighbours of $\mathbf{x}$ in $(X, \delta)$. Then the *normality* of $\mathbf{x}$ in $X$, denoted as $\rho(\mathbf{x})$, is defined as:

$$\rho(\mathbf{x}) = y(\mathbf{x}) \cdot \frac{\sum_{\mathbf{x}' \in N(\mathbf{x}, X)} \delta(\mathbf{x}, \mathbf{x}') \cdot y(\mathbf{x}')}{\sum_{\mathbf{x}' \in N(\mathbf{x}, X)} \delta(\mathbf{x}, \mathbf{x}')} \tag{2}$$

The values of normality range from -1 to 1, indicating the possibility of generalising an error $\mathbf{x}$ into improving a linkage model: (1) $\rho(\mathbf{x}) = -1$ means that all the nearest neighbours in $N(\mathbf{x}, X)$ have the opposite label to $\mathbf{x}$; (2) $\rho(\mathbf{x}) = 1$ means that all the nearest neighbours in $N(\mathbf{x}, X)$ have the same label as $\mathbf{x}$. In addition to the labels, the distances of $\mathbf{x}$ and its nearest neighbours are also taken into account by the normality. For each neighbour in $N(\mathbf{x}, X)$, the close its distance is to $\mathbf{x}$, the more influence it has on $\rho(\mathbf{x})$. Intuitively, the normality of an error $\mathbf{x}$ is measured by a weighted majority vote of their nearest neighbours, i.e., the margin between strength of voting on the same label $y(\mathbf{x})$ and strength of voting on the opposite label $-1 \cdot y(\mathbf{x})$.

*Example 4.2.* Suppose that, for the detected error $\mathbf{x_r}$ in Figure 2, we have $N(\mathbf{x_r}, X_i) = \{\mathbf{x_1}, \ldots, \mathbf{x_6}\}$, and $\delta(\mathbf{x_r}, \mathbf{x_i})$ is 0.9, 0.8, 0.8, 0.8, 0.7 and 0.7 for $i = 1, 2, 3, 4, 5, 6$, respectively. Then $\rho(\mathbf{x_r}) = 1$ in Figure 2.b and $\rho(\mathbf{x_r}) = (0.9 + 0.8 + 0.8 + 0.8 + 0.7 - 0.7)/(0.9 + 0.8 + 0.8 + 0.8 + 0.7 + 0.7) = 0.7$ in Figure 2.c.

### 4.2 Tree-structured Classifiers

To repair linkage models effectively, the repairing framework needs to offer great flexibility and expressive power in refining linkage models. We thus propose to use a tree-based structure for repairing classifiers. A tree-structured classifier can be formalised as a tuple $(T, C, \alpha, \beta)$, where $T$ is a binary tree, $C$ is a set of binary classifiers, $\alpha$ is a labeling function that assigns a classifier $c_i \in C$ to each internal vertex $v_i$ of $T$, i.e., $\alpha(v_i) = c_i$, and $\beta$ assigns a label in $\{-1, 1\}$ to each edge $(v_i, v_j)$ of $T$, i.e., $\beta(v_i, v_j) \in \{-1, 1\}$. Therefore, given a weight vector $\mathbf{x} \in X$, a tree-structured classifier $h = (T, C, \alpha, \beta)$ classifies $\mathbf{x}$ using the following condition: $h(\mathbf{x}) = \beta(v_{n-1}, v_n)$ if there exists a path $\langle v_0, v_1, \ldots, v_n \rangle$ in $T$ starting from the root vertex $v_0$ of $T$ and ending at a leaf vertex $v_n$ such that $\alpha(v_k) = c_k$ and $\bigwedge_{k \in [0, n-1]} c_k(\mathbf{x}) = \beta(v_k, v_{k+1})$.

*Example 4.3.* Consider two weight vectors $\mathbf{x_7}$ and $\mathbf{x_8}$ from the weight vector space depicted in Figure 3.b. The tree-structured classifier in Figure 3.a has classified this weight vector space into many smaller weight vector subspaces, each being assigned a

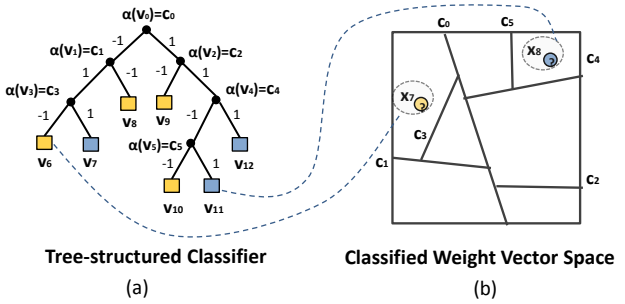


**Figure 3: A schematic layout of a tree-structured classifier and the corresponding classified weight vector space.**

label of either −1 (highlighted by a yellow square) or 1 (highlighted by a blue square). We have $h(\mathbf{x}_7) = -1$ due to a path $\langle v_0, v_1, v_3, v_6 \rangle$ with $c_0(\mathbf{x}_7) = -1, c_1(\mathbf{x}_7) = 1$ and $c_3(\mathbf{x}_7) = -1$. Similarly, we have $h(\mathbf{x}_8) = 1$ because there is a path $\langle v_0, v_2, v_4, v_5, v_{11} \rangle$ with $c_0(\mathbf{x}_8) = 1$, $c_2(\mathbf{x}_8) = 1$, $c_4(\mathbf{x}_8) = -1$, and $c_5(\mathbf{x}_8) = 1$.

Initially, the binary tree $T$ in a tree-structured classifier only has one internal vertex as the root, which represents the initial classifier $c_0$, and two leaf vertices that represent two weight vector subspaces $X_1$ and $X_2$ of the given weight vector space $X$ classified by the initial classifier $c_0$ such that $\forall \mathbf{x}_1 \in X_1(c_0(\mathbf{x}_1) = 1)$ and $\mathbf{x}_2 \in X_2(c_0(\mathbf{x}_2) = -1)$. Then when a linkage error $\mathbf{x}$ is detected, $c$ can be repaired by extending $T$ to $T'$ such that one of the leaf vertices of $T$ (say $v_t$) is replaced by an internal vertex with two leaf vertices. Accordingly, the weight vector subspace associated with $v_t$ is divided into two smaller weight vector subspaces (one with the label 1 and the other with the label -1). This process is iteratively conducted when repairing errors.

As a result, an important property of tree-structured classifiers is to support *bounded repairs*, i.e., only weight vectors in a specified subspace are affected by a repair. This property can reduce the risk of incorrect repairs and lead to convergence on quality of a linkage model after a finite number of repairing iterations.

## 4.3 Repairing Algorithm

Our repairing algorithm is provided in Figure 4. Given a weight vector space $X$ which is partitioned into $\{X_1, \ldots, X_n\}$ by an initial classifier $c_0$, the algorithm iteratively leverages errors to repair a tree-structured classifier until the labelling budget is used up or no more errors are detected (Line 2).

In this algorithm, $E_h$ refers to detected errors of $h$, $q$ keeps record of the number of labels requested from the human oracle, SELECT_ERROR($E_h$) randomly selects an error $e$ from $E_h$, and FIND_SUBSPACE($\mathbf{X}, e$) searches for a subspace $X_{cur}$ of $X$ which contains $e$ (Lines 3-4). If $|X_{cur}|$ is greater than $S_{min}$ (Line 5), then NN_SELECT($e, X_{cur}, k$) finds $k$ nearest neighbours of $e$ in $X_{cur}$, and their labels are requested (Lines 6-7). If the normality $\rho(e)$ of $e$ is greater than the threshold $t$ (Line 9), FN_SELECT($e, X_{cur}, k$) finds $k$ farthest neighbours of $e$ and their labels are requested (Lines 10-11). If the purity of $X_{cur}$ is lower than the threshold $p$ (Line 13), the classifier $c_{q+1}$ is trained and extends $c_q$ by classifying $X_{cur}$ into two subspaces $X^M_{cur}$ (for matches) and $X^N_{cur}$ (for non-matches) (Lines 14-16). Then accordingly, the tree-structured classifier $h$ is repaired by adding $c_{q+1}$ (Line 17). We consider errors whose normality values are lower than the threshold them as outliers. The algorithm returns a tree-structured classifier $h$ with improved linkage quality (Line 23).

*Input:* initial classifier: $c_0$, normality threshold: $t \in [-1.1]$, human oracle for labelling: ORACLE(), budget limit: $b$, number of nearest neighbours: $k$, purity threshold $p$, minimum size of weight vector subspaces: $s_{min}$
*Output:* a tree-structured classifier $h$

1: $\mathbf{X} \leftarrow \{X_1, \ldots, X_n\}, q \leftarrow 1, h \leftarrow c_0$
2: **while** $q \leq b$ and $|E_h| > 0$ **do**:
3: $e \leftarrow$ SELECT_ERROR($E_h$)
4: $X_{cur} \leftarrow$ FIND_SUBSPACE($\mathbf{X}, e$)
5: **if** $|X_{cur}| \geq s_{min}$ **then**:
6: $V_{nn} \leftarrow$ NN_SELECT($e, X_{cur}, k$)
7: $V^L_{nn} \leftarrow$ ORACLE($V_{nn}$)
8: $V^L \leftarrow V^L \cup V^L_{nn}$
9: **if** $\rho(e) \geq t$ **then**:
10: $V_{fn} \leftarrow$ FN_SELECT($e, X_{cur}, k$)
11: $V^L_{fn} \leftarrow$ ORACLE($V_{fn}$)
12: $V^L \leftarrow V^L \cup V^L_{fn}$
13: **if** $purity(X_{cur}) < p$ **then**
14: $c_{q+1} \leftarrow$ TRAIN($c_q, X_{cur}, V^L$)
15: $X^M_{cur}, X^N_{cur} \leftarrow c_{q+1}$.CLASSIFY($X_{cur}$)
16: $\mathbf{X} \leftarrow \mathbf{X} - \{X_{cur}\} \cup \{X^M_{cur}, X^N_{cur}\}$
17: $h =$ REPAIR_CLASSIFIER($h, c_{q+1}$)
18: **endif**:
19: **endif**:
20: **endif**:
21: $q \leftarrow q + 1$
22: **endwhile**:
23: **return** $h$

**Figure 4: Repairing Algorithm**

# 5 EXPERIMENTS

We have implemented the repairing framework to experimentally validate its effectiveness on three real-world datasets.

**Baseline methods.** We have used support vector machines (SVM) with three different kernels: linear, polynomial, and Gaussian (RBF) and a decision tree as the baseline methods. For our tree-structured classifier, we used SVMs with linear kernel as binary classifiers for its internal vertices, called SVM-tree. We also used the following parameters: $s_{min} = 25$, $p = 0.97$, and $k = 50$.

**Sampling methods.** We use KNN+FNN to refer to the $k$ nearest and farthest neighbours sampling method described in the repairing algorithm in Figure 4. To evaluate its effectiveness, we compared it with three other sampling methods: (1) KNN, (2) FNN, and (3) random, in which k nearest, k farthest, and k random neighbours of an error are sampled, respectively.

**Datasets.** We have used three datasets: (1) Cora, which contains 1,295 publication records and is publicly available [1], (2) DBLP_ACM, which contains 4910 bibliographic records from the DBLP and ACM websites [8], and (3) NCVR, using two datasets with 1048576 and 613767 records respectively from the North Carolina Voter Registration (NCVR) database[2]. We did not apply any blocking on Cora but used *publication year* and the 1st char of *title* for DBLP_ACM, and *last_name* and *first_name* for NCVR.

**Performance of classifiers.** Figure 5 shows the performance of SVM-tree in comparison to the baseline methods. SVM-tree

[1] Available from: `http://secondstring.sourceforge.net`
[2] Available from: `ftp://alt.ncsbe.gov/data/`

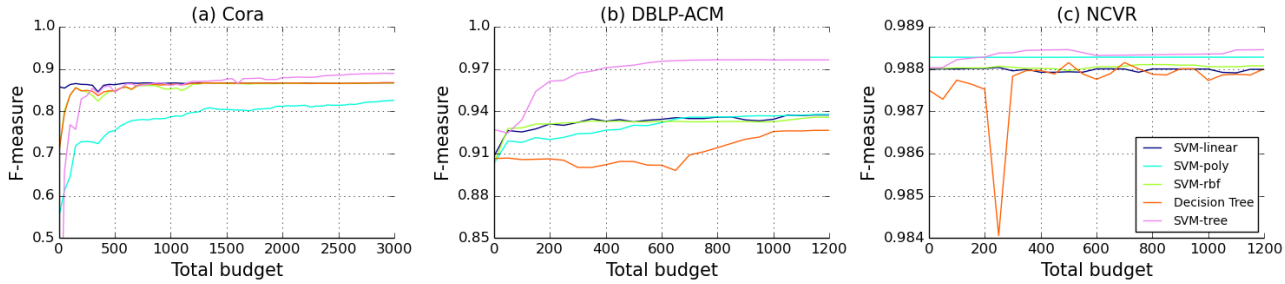


**Figure 5: Comparison to the baseline methods, where the sampling method is KNN+FNN and normality threshold is -0.6.**

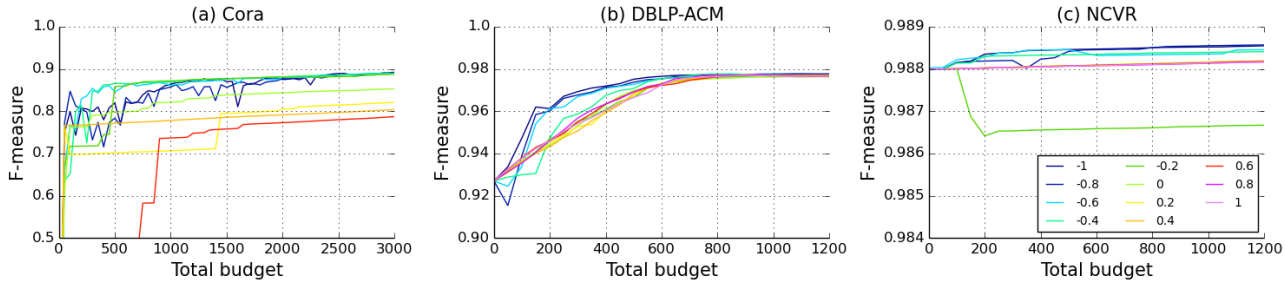


**Figure 6: SVM-tree with different normality thresholds, where the sampling method is KNN+FNN.**

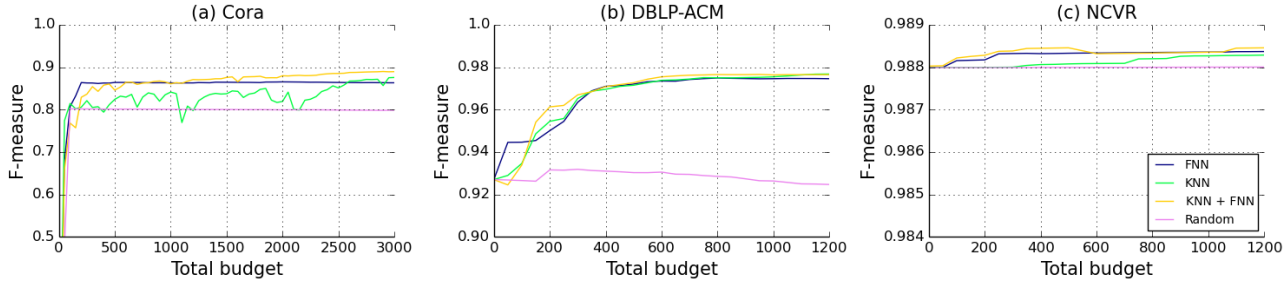


**Figure 7: SVM-tree with different sampling methods, where the normality threshold is -0.6.**

generally performs better than the other methods for all datasets. For Cora, SVM-tree performs increasingly better when more errors are detected. For DBLP_ACM, it has a significant increase in performance. For NCVR, it is a clean dataset that produces high quality results (around 0.988 f-measure) for all methods; nonetheless, SVM-tree is still marginally better.

**Effect of normality.** Figure 6 presents the experimental results for classification using SVM-tree with varying thresholds. The KNN+FNN sampling method is used in this experiment. A normality threshold of -0.4 to -1 produces the best performance for all datasets. The threshold of -0.6 produces consistently stable good results over all datasets.

**Performance of sampling.** Figure 7 shows the performance of SVM-tree using different sampling methods and the normality threshold is set to -0.6. KNN+FNN generally produces the best and most stable results among all the sampling methods, particularly over the datasets Cora and DBLP_ACM. When the budget is low, the performance of FNN is comparable to KNN+FNN. The performance of KNN is not stable on Cora. The random sampling has the worst performance among all the methods.

## ACKNOWLEDGEMENT

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

## REFERENCES

[1] Foto N Afrati and Phokion G Kolaitis. 2009. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*. 31–41.
[2] I. Bhattacharya and L. Getoor. 2007. Collective entity resolution in relational data. *TKDD* 1, 1 (2007), 5.
[3] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving data quality: Consistency and accuracy. In *PVLDB*. 315–326.
[4] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate Record Detection: A Survey. *IEEE TKDE* 19 (2007), 1–16.
[5] I.P. Fellegi and A.B. Sunter. 1969. A theory for record linkage. *J. Amer. Statistical Assoc.* 64, 328 (1969), 1183–1210.
[6] Jeffrey Fisher, Peter Christen, and Qing Wang. 2016. Active learning based entity resolution using Markov logic. In *PAKDD*. 338–349.
[7] Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. 2015. A Clustering-Based Framework to Control Block Sizes for Entity Resolution. In *KDD*. 279–288.
[8] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *VLDB* 3, 1-2 (2010), 484–493.
[9] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *VLDB* 9, 9 (2016), 684–695.
[10] Jingyu Shao and Qing Wang. 2018. Active Blocking Scheme Learning for Entity Resolution. In *PAKDD*. 350–362.
[11] Qing Wang, Mingyuan Cui, and Huizhi Liang. 2016. Semantic-Aware Blocking for Entity Resolution. *TKDE* 28, 1 (2016), 166–180.
[12] Qing Wang, Dinusha Vatsalan, and Peter Christen. 2015. Efficient Interactive Training Selection for Large-Scale Entity Resolution. In *PAKDD*. 562–573.
[13] Steven Euijong Whang and Hector Garcia-Molina. 2010. Entity resolution with evolving rules. *VLDB* 3, 1-2 (2010), 1326–1337.
[14] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. 2006. *Similarity search: the metric space approach*. Vol. 32. Springer.