Technical Report AI-02-296, Artificial Intelligence Lab,
University of Texas at Austin, February 2002.

Learning to Combine Trained Distance Metrics for Duplicate Detection in Databases *

Mikhail Bilenko and Raymond J. Mooney
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
{mbilenko,mooney}@cs.utexas.edu

February 22, 2002

Abstract

The problem of identifying approximately duplicate records in databases has previously been studied as record linkage, the merge/purge problem, hardening soft databases, and field matching. Most existing approaches have focused on efficient algorithms for locating potential duplicates rather than precise similarity metrics for comparing records. In this paper, we present a domain-independent method for improving duplicate detection accuracy using machine learning. First, trainable distance metrics are learned for each field, adapting to the specific notion of similarity that is appropriate for the field's domain. Second, a classifier is employed that uses several diverse metrics for each field as distance features and classifies pairs of records as duplicates or non-duplicates. We also propose an extended model of learnable string distance which improves over an existing approach. Experimental results on real and synthetic datasets show that our method outperforms traditional techniques.

Keywords: *data cleaning, deduplication, record linkage, distance metric learning, trained similarity measures, string edit distance*

1 Introduction

Databases frequently contain approximately duplicate field-values and records that refer to the same entity but are not identical. Variations in representation can arise from typographical errors, misspellings, abbreviations, as well as other sources. Variations are particularly pronounced in data that is automatically extracted from unstructured or semi-structured documents or web pages [15, 8, 3]. Such variant duplicates can have many deleterious effects, including preventing data-mining algorithms from discovering important regularities. Such problems are typically handled during a tedious manual "data cleaning" or "de-duping" process.

Some previous work has addressed the problem of identifying duplicate records, where it is referred to as record linkage [17, 25], the merge/purge problem [10], duplicate detection [14], hardening soft databases [3], and reference matching [12]. Typically, a fixed textual similarity metric such as edit distance [22] or vector-space cosine similarity [21] is used to determine whether two values or records are alike enough to be duplicates.

*This research was supported by the National Science Foundation under grant IIS-0117308.

However, the similarity of two strings can depend on the domain and field under consideration. For example, deleting the substring "Street" may be acceptable when comparing addresses but not when comparing names of people (e.g. "Nick Street"), web sites (e.g. "TheStreet.com"), or newspapers (e.g. "Wall Street Journal"). Rather than hand-tuning a distance metric for each field in each domain, we present a method for automatically learning an appropriate string-similarity metric from small corpora of hand-labeled examples. When computing edit distance, a different cost can be assigned to each edit operation. These costs are typically set manually; however, an algorithm was recently introduced for learning appropriate costs by training on a set of labeled examples [20]. We consider an extended model of edit distance and employ a similar Expectation Maximization (EM) method to train metrics appropriate for each field.

Different types of textual similarity, such as "bag of words" metrics versus string-based edit distances, have complementary strengths and weaknesses. Consequently, it is also useful to consider multiple similarity metrics when evaluating potential duplicates. The utility of different metrics is task-dependent, and therefore it is also preferable to adaptively learn an appropriate function for combining them [2]. In our approach, Support Vector Machines (SVM's) [23], are used to learn a function of multiple similarity metrics that best discriminates duplicates from non-duplicates.

Our overall system, MARLIN (Multiply Adaptive Record Linkage with INduction), employs a two-level learning approach. First, a set of similarity metrics are trained to appropriately determine the similarity of different field values. Next, a final predicate for detecting duplicate records is learned from multiple (trained and static) similarity metrics applied to each of the individual fields. Experimental results on real and synthetic datasets show that MARLIN is more accurate than traditional techniques.

2 Learnable String Distance

2.1 Background

There are several well-known methods for estimating similarity between strings, which can be roughly separated into two groups: token-based techniques and character-based techniques. Jaccard and vector-space cosine similarity are examples of difference measures that operate on tokens, treating a string as a "bag of words" [21]. Character-based measures, such as Levenshtein distance and its variants [9], compute similarity between strings by estimating the minimum sequence of changes that transform one string into another. When data is represented by relatively short strings that contain similar yet orthographically distinct tokens, character-based measures are preferable since they can estimate the difference between the strings with higher resolution.

Original string edit distance proposed by Levenshtein [11] is defined as the minimum number of insertions, deletions or substitutions necessary to transform one string into another. Needleman and Wunsch [16] extended the distance model to allow contiguous sequences of mismatched characters, or gaps, in the alignment of two strings and described a general dynamic programming method for computing edit distance. Most commonly the gap penalty is calculated using the *affine* model: $cost(g) = s + e \times l$, where s is the cost of opening a gap, e is the cost of extending a gap, and l is the length of a gap in the alignment of two strings, assuming that all characters have a unit cost. Usually e is set to a value lower than s, thus decreasing the penalty for contiguous mismatched substrings. Since differences between duplicate records often arise because of abbreviations or whole-word insertions and deletions, this model produces a more sensitive similarity estimate than Levenshtein distance.

String distance with affine gaps, $S(x^T, y^V)$, between strings x^T of length T and y^V of length V, can be computed using a dynamic programming algorithm that constructs three matrices based on the following recurrences in $O(TV)$ computational time:

$$M_{i,j} = \min \begin{cases} M_{i-1,j-1} + c(x_i, y_j) \\ I_{i-1,j-1} + c(x_i, y_j) \\ D_{i-1,j-1} + c(x_i, y_j) \end{cases}$$

$$D_{i,j} = \min \begin{cases} M_{i-1,j} + s + c(x_i, \epsilon) \\ D_{i-1,j} + e + c(x_i, \epsilon) \end{cases} \tag{1}$$

$$I_{i,j} = \min \begin{cases} M_{i,j-1} + s + c(\epsilon, y_j) \\ I_{i,j-1} + e + c(\epsilon, y_j) \end{cases}$$

$$S(x^T, y^V) = \min(I_{T,V}, D_{T,V}, M_{T,V})$$

Each matrix element $M_{i,j}$ contains the distance between substrings $x_{0\ldots i}$ and $y_{0\ldots j}$ for an alignment where the last two characters of the substrings, x_i and y_j, are aligned, while matrix elements $I_{i,j}$ and $D_{i,j}$ give the distances between substring alignments that end in insertion and deletion gaps respectively. Cost of a single edit operation (insertion, deletion or substitution) that aligns character x_i to character y_j is given by $c(x_i, y_j)$, where either x_i or y_j can be the null string ϵ, corresponding to a part of a gap. The final distance between the strings is the minimum of three alignments: $M_{T,V}$, matching the last two characters of the two strings; $D_{T,V}$, matching the last character of the first string with a gap in the second string; and $I_{T,V}$, matching the last character of the second string with a gap in the first string.

2.2 Learnable Distance Metrics

Different edit operations have varying significance in different domains. For example, inserting, deleting and substituting symbols "-" and "/" is common and unimportant when phone numbers are being matched, while any edit operation involving digits introduces a semantically meaningful difference. Therefore, adapting a string similarity metric to a particular domain requires assigning different weights $c(x_i, y_j)$ to different edit operations, a task that has traditionally been performed manually either using domain knowledge or by trial and error.

Ristad and Yianilos [20] proposed a generative model of Levenshtein distance along with an Expectation-Maximization algorithm that learns model parameters using a training corpus of matched strings. In their model, a string alignment is equivalent to a sequence of character pairs generated by edit operations emitted by a memoryless stochastic transducer. Each edit operation corresponds to the probability of producing a substitution $p(\langle x_i, y_j \rangle)$, an insertion $p(\langle \epsilon, y_j \rangle)$, or a deletion $p(\langle x_i, \epsilon \rangle)$, where probabilities of all operations are normalized.

An analogous generative model can be constructed for string distance with affine gaps. Given an alphabet of symbols $A^* = A \bigcup \{\epsilon\}$, the full set of edit operations is $E = E_s \bigcup E_d \bigcup E_i$, where $E_s = \{\langle a, b \rangle \mid a, b \in A\}$ is the set of all substitution and matching operations $\langle a, b \rangle$; and $E_i = \{\langle \epsilon, a \rangle \mid a \in A\}$ and $E_d = \{\langle a, \epsilon \rangle \mid a \in A\}$ are sets of insertion and deletion operations respectively.

Each of the three matrices of the original affine gap model (1) corresponds to one of the states of the generative model in Fig.1. A pair of matched strings is generated by this model as a sequence of traversals along the edges accompanied by emissions of characters pairs, which are determined by the state that is reached via each traversal. The production starts in state M and terminates when special state $\#$ is reached.

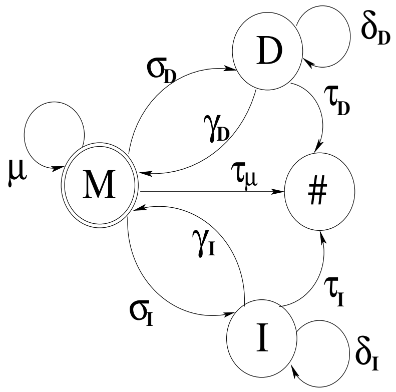

Figure 1: Generative model for string distance with affine gaps

Edit operations emitted in each state correspond to aligned pairs of characters: substitutions $\langle a, b\rangle$ and exact matches $\langle a, a\rangle$ in state M; deletions from the first string $\langle a, \epsilon\rangle$ in state D; and insertions of characters from the second string into the first string $\langle \epsilon, a\rangle$ in state I. Each edit operation $e \in E$ is assigned a probability $p(e)$ such that $\sum_{e\in E_s} p(e) = 1$, $\sum_{e\in E_d} p(e) = 1$, and $\sum_{e\in E_i} p(e) = 1$. Edit operations with higher probabilities produce character pairs that are likely to be aligned in a given domain, such as substitution ("/", "-") for phone numbers, or deletion (".", ϵ) for addresses.

Transitions σ_D and σ_I from the matching state M to either the deletion state D or the insertion state I correspond to starting a gap in one of the strings. A gap is ended when edges γ_D and γ_I are traversed back to the matching state. Remaining in state M by taking edge μ corresponds to a sequence of substitutions or exact matches of characters, while remaining in states I and D is analogous to extending a gap in either the first or the second string. Thus, gap opening and extension costs s and e of the original affine model (1) correspond to transition probabilities σ_D, σ_I, δ_D and δ_I. The sum of transition probabilities must be normalized in each state for the model to be complete:

$$\begin{aligned}\mu + \sigma_D + \sigma_I + \tau_\mu = 1\\ \delta_D + \gamma_D + \tau_D = 1\\ \delta_I + \gamma_I + \tau_I = 1\end{aligned} \quad (2)$$

This generative model is similar to one given for amino-acid sequences in [6] with two differences: (1) transition probabilities are distinct for states D and I, and (2) every transition has a probability parameter associated with it, instead of being expressed through other transitions outgoing from the same state.

Given two strings, x^T of length T and y^V of length V, we can calculate probabilities of generating the pair of prefixes $(x^T_{1\ldots t}, y^V_{1\ldots v})$ and suffixes $(x^T_{t+1\ldots T}, y^V_{v+1\ldots V})$ using dynamic programming in forward and backward algorithms shown in Fig.2 and Fig.3 in $O(TV)$ time. The total probability of generating a pair of strings $p(x^T, y^V)$ by taking all possible paths through the model is the same for forward and backward algorithms.

Given a corpus of n matched strings corresponding to pairs of duplicates, $C = \{(x^{T_1}, y^{V_1}), \ldots, (x^{T_n}, y^{V_n})\}$, this model can be trained using the Baum-Welch algorithm, which is a variant of the Expectation-Maximization procedure for learning parameters of generative models [19], shown in Fig.4. The training procedure iterates between two steps, shown in Fig.5 and Fig.6. In each EXPECTATION-STEP, the expected number of occurrences for each state transition and edit operation emission is accumulated for a given pair of strings (x^T, y^V) from the training corpus. This is achieved by accumulating the posterior probabilities for every possible state transition and an accompanying emission in lines 7-20. In the MAXIMIZATION procedure all model parameters are updated using the collected expectations.

FORWARD(x^T, y^V)

$M_{0,0} = 1;\;\; D_{0,0} = 0;\;\; I_{0,0} = 0$
for $i = 0$ to T
  for $j = 0$ to V
    if $(i > 0)$
      $D_{i,j} = p(\langle x_i, \epsilon\rangle)[\sigma_D M_{i-1,j} + \delta_D D_{i-1,j}]$
    if $(j > 0)$
      $I_{i,j} = p(\langle \epsilon, y_j\rangle)[\sigma_I M_{i,j-1} + \delta_I I_{i,j-1}]$
    if $(i > 0 \wedge j > 0)$
      $M_{i,j} = p(\langle x_i, y_j\rangle)[\mu M_{i-1,j-1} +$
        $+ \gamma_I I_{i-1,j-1} + \gamma_D D_{i-1,j-1}]$
$p(x^T, y^V) = \tau_\mu M_{T,V} + \tau_D D_{T,V} + \tau_I I_{T,V}$
return M, I, D,$p(x^T, y^V)$

Figure 2: Forward algorithm for generative string distance with affine gaps

BACKWARD(x^T, y^V)

$M_{T,V} = \tau_\mu;\;\; D_{T,V} = \tau_D;\;\; I_{T,V} = \tau_I$
for $i = T$ downto 0
  for $j = V$ downto 0
    if $(i < T)$
      $D_{i,j} = p(\langle x_{i+1}, \epsilon\rangle)\delta_D D_{i+1,j}$
      $M_{i,j} = p(\langle x_{i+1}, \epsilon\rangle)\gamma_D D_{i+1,j}$
    if $(j < V)$
      $I_{i,j} = p(\langle \epsilon, y_{j+1}\rangle)\delta_I I_{i,j+1}$
      $M_{i,j} \mathrel{+}= p(\langle \epsilon, y_{j+1}\rangle)\gamma_I I_{i,j+1}$
    if $(i < T \wedge j < V)$
      $D_{i,j} \mathrel{+}= p(\langle x_{i+1}, y_{j+1}\rangle)\sigma_D M_{i+1,j+1}$
      $I_{i,j} \mathrel{+}= p(\langle x_{i+1}, y_{j+1}\rangle)\sigma_I M_{i+1,j+1}$
      $M_{i,j} \mathrel{+}= p(\langle x_{i+1}, y_{j+1}\rangle)\mu M_{i+1,j+1}$
$p(x^T, y^V) = M_{0,0}$
return M, I, D,$p(x^T, y^V)$

Figure 3: Backward algorithm for generative string distance with affine gaps

EXPECTATION-MAXIMIZATION($\{(x^{T_1}, y^{V_1}), \ldots, (x^{T_n}, y^{V_n})\}$)
 until convergence
  for $i = 0$ to n
   EXPECTATION-STEP((x^{T_i}, y^{V_i}))
  MAXIMIZATION()

Figure 4: Training algorithm for generative string distance with affine gaps

EXPECTATION-STEP((x^T, y^V))

$(M^f, I^f, D^f, p(x^T, y^V)) = $ FORWARD$(x^T, y^V))$

$(M^b, I^b, D^b, p(x^T, y^V)) = $ BACKWARD$(x^T, y^V))$

$E(\langle \tau_D \rangle) \mathrel{+}= 1;\ E(\langle \tau_I \rangle) \mathrel{+}= 1$

$E(\langle \tau_\mu \rangle) \mathrel{+}= 1$

for $i = 1$ to T

  for $j = 1$ to V

    $\xi_\mu = \frac{M^f_{j-1,k-1} * \mu * p(\langle x_i, y_j \rangle) * M^b_{j,k}}{p(x^T, y^V)}$

    $E[\mu] \mathrel{+}= \xi_\mu;\ E[\langle x_i, y_j \rangle) \mathrel{+}= \xi_\mu$

    $\xi_{\sigma_I} = \frac{M^f_{j,k-1} * \sigma_I * p(\langle \epsilon, y_j \rangle) * I^b_{j,k}}{p(x^T, y^V)}$

    $E[\sigma_I] \mathrel{+}= \xi_{\sigma_I};\ E[\langle x_i, \epsilon \rangle] \mathrel{+}= \xi_{\sigma_I}$

    $\xi_{\sigma_D} = \frac{M^f_{j-1,k} * \sigma_D * p(\langle x_i, \epsilon \rangle) * D^b_{j,k}}{p(x^T, y^V)}$

    $E[\sigma_D] \mathrel{+}= \xi_{\sigma_D};\ E[\langle \epsilon, y_j \rangle] \mathrel{+}= \xi_{\sigma_D}$

    $\xi_{\delta_I} = \frac{I^f_{j,k-1} * \delta_I * p(\langle \epsilon, y_j \rangle) * I^b_{j,k}}{p(x^T, y^V)}$

    $E[\delta_I] \mathrel{+}= \xi_{\delta_I};\ E[\langle \epsilon, y_j \rangle] \mathrel{+}= \xi_{\delta_I}$

    $\xi_{\delta_D} = \frac{D^f_{j-1,k} * \delta_D * p(\langle x_j, \epsilon \rangle) * D^b_{j,k}}{p(x^T, y^V)}$

    $E[\delta_D] \mathrel{+}= \xi_{\delta_D};\ E[\langle x_i, \epsilon \rangle] \mathrel{+}= \xi_{\delta_D}$

    $\xi_{\gamma_I} = \frac{I^f_{j-1,k-1} * \gamma_D * p(\langle x_i, y_j \rangle) * M^b_{j,k}}{p(x^T, y^V)}$

    $E[\gamma_I] \mathrel{+}= \xi_{\gamma_I};\ E[\langle x_i, y_j \rangle] \mathrel{+}= \xi_{\gamma_I}$

    $\xi_{\gamma_D} = \frac{D^f_{j-1,k-1} * \gamma_D * p(\langle x_i, y_j \rangle) * M^b_{j,k}}{p(x^T, y^V)}$

    $E[\gamma_D] \mathrel{+}= \xi_{\gamma_D};\ E[\langle x_i, y_j \rangle] \mathrel{+}= \xi_{\gamma_D}$

Figure 5: Expectation step for generative string distance with affine gaps

It can be proved that this training procedure is guaranteed to converge to a local maximum of likelihood of observing the training corpus C. The trained model can be used for estimating distance between two strings by computing the probability of generating the aligned pair of strings summed across all possible paths as calculated by the FORWARD and BACKWARD algorithms: $d(x^T, y^V) = -\log p(x^T, y^V)$. A practical problem that may arise in this computation is numerical underflow for long strings, which can be solved by mapping all computations into logarithmic space or by periodic scaling of all values in matrices M, D and I [20].

2.3 Adapting Learned String Distance with Affine Gaps for Duplicate Detection

Because the order of strings being aligned does not matter when similarity of database records is being estimated, insertion and deletion operations as well as transitions for states I and D can be represented by a single set of parameters: $p(\langle a, \epsilon \rangle) = p(\langle \epsilon, a \rangle)$ for all symbols $a \in A$; $\tau = \tau_I = \tau_D; \gamma = \gamma_I = \gamma_D; \delta = \delta_I = \delta_D; \sigma = \sigma_I = \sigma_D$. All algorithms described above then would use the unified set of parameters instead of separate sets of values for states I and D.

The generative model of Fig.1 suffers from two drawbacks that impede its utility for computing similarity between strings in a database. One problem lies in the fact that the model assigns a probability less than

MAXIMIZATION()

1. $N_M = E[\mu] + E[\sigma_I] + E[\sigma_D] + E[\tau_\mu]$
2. $\mu = E[\mu]/N_M; \sigma = E[\sigma]/N_M; \tau_\mu = E[\tau_\mu]/N_M$
3. $N_I = E[\delta_I] + E[\gamma_I] + E[\tau_{\delta_I}]$
4. $\delta_I = E[\delta_I]/N_I; \gamma = E[\gamma_I]/N_I; \tau_{\delta_I} = E[\tau_{\delta_I}]/N_I$
5. $N_D = E[\delta_D] + E[\gamma_D] + E[\tau_{\delta_D}]$
6. $\delta_D = E[\delta_D]/N_D; \gamma = E[\gamma_D]/N_D; \tau_{\delta_D} = E[\tau_{\delta_D}]/N_D$
7. for each $\langle a, b\rangle$
8. $\quad N'_M \mathrel{+}= E[\langle a, b\rangle]$
9. for each $\langle \epsilon, a\rangle$
10. $\quad N'_I \mathrel{+}= E[\langle \epsilon, a\rangle]$
11. for each $\langle a, \epsilon\rangle$
12. $\quad N'_D \mathrel{+}= E[\langle a, \epsilon\rangle]$
13. for each $\langle a, b\rangle$
14. $\quad p(\langle a, b\rangle) = E[\langle a, b\rangle]/N'_M$
15. for each $\langle \epsilon, a\rangle$
16. $\quad p(\langle \epsilon, a\rangle) = E[\langle \epsilon, a\rangle]/N'_I$
17. for each $\langle a, \epsilon\rangle$
18. $\quad p(\langle a, \epsilon\rangle) = E[\langle a, \epsilon\rangle]/N'_D$

Figure 6: Maximization step for generative string distance with affine gaps

one to strings that are exact duplicates. Because the probability of an alignment monotonically decreases as more matching characters are appended to the strings, longer exact duplicates are penalized even more severely than shorter exact duplicates, which is counter-intuitive and exacerbates the problem further.

The second difficulty lies in the fact that due to the large size of the edit operation set, probabilities of individual operations are significantly smaller than transition probabilities. If only a relatively small number of training examples is available, probabilities of some edit operations may be underestimated, and distances assigned to strings will vary significantly with minor character variations. There are two steps that need to be taken to address these issues. First, the probability distribution over the set of edit operations, E, is smoothed by bounding each edit operation probability by some minimum value λ. This is achieved by adding λ to each updated probability in lines 11-16 of the MAXIMIZATION procedure and subsequent normalization. Second, learned parameters of the generative distance model are mapped to operation costs of the additive model (1) by taking the negative logarithm of each probability. Distance can then be calculated analogously to Eq.(1) with the addition of supplemental costs $g = -\log\gamma$ for ending a gap and $m = -\log\mu$ for continuing to substitute/match characters. This is equivalent to calculating the cost of the most likely (Viterbi) alignment of the two strings by the generative model in log-space. To solve the "non-zero exact match" problem and decrease high variance in distances due to edit operation costs $c(a, b)$ compared to transition costs s, e, g and m, we dynamically scale edit operation costs to values between 0 and the cost of the state transition that precedes emitting the operation. We also scale the overall distance by the length of the larger string to correct for the "increasing distance for longer exact duplicates" problem. Thus, the resulting metric can be viewed as a hybrid between the generative model and the original fixed-cost model.

3 Record-level Similarity

3.1 Combining similarity across multiple fields

When the distance between records composed of multiple fields is being calculated, it is necessary to combine similarity estimates for individual fields in a meaningful manner. Because correspondence between overall record similarity and similarity across individual fields can vary greatly, it is necessary to weight fields according to their contribution to the true similarity between records.

While statistical aspects of combining similarity scores for individual fields have been addressed in previous work on record linkage [25], availability of labeled duplicates allows a more direct approach that uses a binary classifier [2]. Given a database that contains records composed of k different fields and a set $D = \{d_1, \ldots, d_m\}$ of distance metrics, we can represent any pair of records by an mk-dimensional vector of "distance features". Each component of the vector represents similarity between two fields of the records calculated using one of the distance metrics. Matched pairs of duplicate records $R = \{(r_{10}, r_{11}), \ldots, (r_{n0}, r_{n1})\}$ can be used to construct a training set of such vectors by assigning them a positive class label. Pairs of records that are not labeled as duplicates form the complementary set of negative examples.

A binary classifier can then be trained using these vectors to discriminate between pairs of records corresponding to duplicates and non-duplicates. Such a classifier must possess several important properties:

1. Resilience to irrelevant features that correspond to those fields that do not carry discriminatory information.

2. Ability to handle correlated features corresponding to fields that are closely related.

3. Independence from the relative sizes of the positive and negative example sets. Because it could be easier for a human expert to identify a small subset of duplicate records, achieving equal proportions of duplicate and non-duplicate records in the training set and the testing set could be difficult.

4. Capability to compute confidence estimates for class membership.

Support vector machines [23] satisfy all of these requirements. SVM's perform binary classification based on the concept of structural risk minimization by mapping data into a high-dimensional space where the two classes are separated by a hyperplane via a kernel function. SVM's exhibit remarkable resistance to noise, handle correlated features well, and rely only on most informative training examples, which implies independence from the relative sizes of the sets of positive and negative examples. Confidence estimates of belonging to each class are naturally given by a datapoint's distance from the hyperplane that separates classes of duplicates and non-duplicates in high-dimensional space.

3.2 Duplicate detection algorithm

A confidence estimate of belonging to the class of duplicates for a given pair of records can be viewed as an overall measure of similarity between the records comprising the pair. Given a large database, producing all possible pairs of records and computing similarity between them is too expensive since it would require $\frac{n^2-1}{2}$ distance computations. The sorted neighborhood method [10] dramatically reduces the number of potential duplicate pairs. The database is sorted using different fields as keys in several passes. After each sorting pass, potential pairs of duplicates are generated by sliding a window of fixed size over the sorted database and adding all pairs of records that co-occur within the window. This technique generates $O(wN)$ pairs, where w is the window size and N is the total number of records in the database.

The canopies clustering method [12] is an alternative approach to reducing the number of candidate pairs. It utilizes some computationally inexpensive metric d_c, such as Jaccard similarity based on an inverted index, to separate records into overlapping clusters ("canopies") of potential duplicates. Jaccard similarity between two strings s_1 and s_2 composed of tokens $\{s_{10}, \ldots, s_{1v}\}$ and $\{s_{20}, \ldots, s_{2w}\}$ is given by:

$$J(s_1, s_2) = \frac{|\{s_{10}, \ldots, s_{1v}\} \bigcap \{s_{20}, \ldots, s_{2w}\}|}{|\{s_{10}, \ldots, s_{1v}\} \bigcup \{s_{20}, \ldots, s_{2w}\}|} \tag{3}$$

The clustering process requires two distance thresholds: T_{loose} for the maximum distance between records that are placed in the same cluster and correspond to potential duplicates, and T_{tight} for limiting the number of clusters by considering only records at a distance of at least T_{tight} from each other as possible cluster centers, where $T_{loose} > T_{tight}$. Candidate pairs are then equivalent to all pairs of records that fall in each cluster, where clusters may overlap.

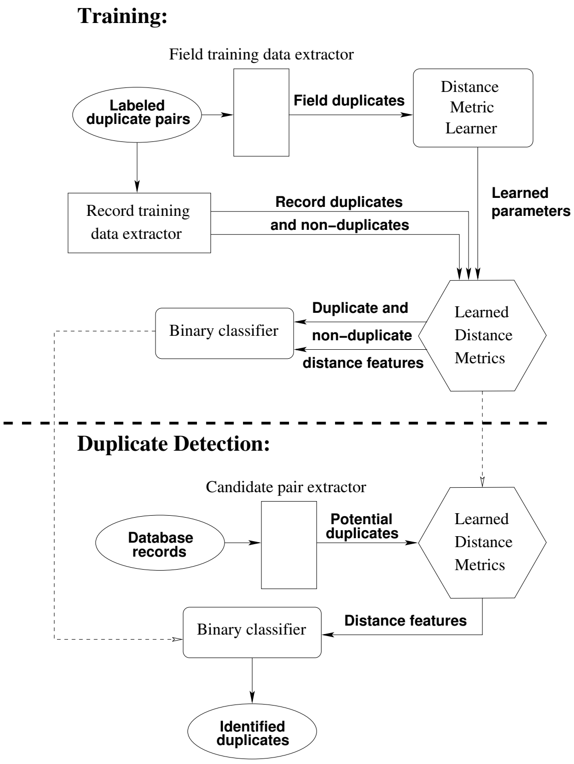

Figure 7: MARLIN overview

An overall view of our system, MARLIN, is presented in Fig.7. The training phase consists of two steps. First, the learnable distance metrics are trained for each record field. The training corpus of paired field-level duplicates is obtained by taking pairs of values for each field from the set of paired duplicate records. Because duplicate records may contain individual fields that are not equivalent, training data can be noisy. This does not pose a serious problem for our approach, since particularly noisy fields that are unhelpful for identifying record-level duplicates will be ignored by the binary classifier as irrelevant distance features.

After individual similarity metrics are learned, they are used to compute distances for each field of duplicate and non-duplicate record pairs to obtain training data for the binary classifier in the form of vectors composed of distance features. For a given training set that contains n duplicate pairs, $O(n^2)$ non-duplicate pairs can be generated. Because we are employing a classifier that does not depend on the relative sizes of training data for the two classes, it is sufficient to randomly add n non-duplicate record pairs to the training set.

The duplicate detection phase starts with the generation of potential duplicate pairs using either the sorted neighborhood or canopies method. This process requires selecting parameter values for either the window size w or for the canopy thresholds T_{loose} and T_{tight}. This can be done by applying the chosen method to the training data and selecting parameter values that result in labeling all true duplicate pairs as candidates.

Next, learned distance metrics are used to calculate distances for each field of each pair of potential duplicate records, thus creating distance feature vectors for the classifier. Confidence estimates for belonging to the class of duplicates are then produced by the binary classifier for each candidate pair, and pairs are sorted by increasing confidence.

The problem of finding a similarity threshold for separating duplicates from non-duplicates arises at this point. A trivial solution would be to use the binary classification results to label some records as duplicates, and others as non-duplicates. A traditional approach to this problem [25], however, requires assigning two thresholds: one that separates pairs of records that are high-confidence duplicates, and another for possible duplicates that should be reviewed by a human expert. Since relative costs of labeling a non-duplicate as a duplicate (false positives) and overlooking true duplicates (false negatives) can vary from database to database, there is no "silver bullet" solution to this problem. Availability of labeled data, however, allows us to provide precision-recall estimates for any threshold value and thus offer a way to control the trade-off between false and unidentified duplicates by selecting threshold values that are appropriate for a particular database.

It is highly likely that several identified duplicate pairs will contain the same record. Since the "duplicate of" relation is transitive, it is necessary to compute the transitive closure of equivalent pairs to complete the identification process. Following [14], we utilize the union-find data structure to store all database records in this step, which allows updating the transitive closure of identified duplicates incrementally in an efficient manner.

4 Experimental Evaluation

4.1 Datasets

We have used three different datasets for our experiments. RESTAURANT is a database of 864 restaurant names and addresses containing 112 duplicates assembled by Sheila Tejada from Fodor's and Zagat's guidebooks. The second dataset, CORA, is a collection of 1295 distinct references to 122 Computer Science research papers from the Cora Computer Science research paper search engine[1]. Finally, we used the database generator of Hernández and Stolfo [10] that randomly corrupts records to introduce duplicates into a mailing list database to create the MAILING dataset of 1200 records corresponding to 400 original entries. Tables 1–3 contain sample duplicate records from each of the databases.

4.2 Experimental Methodology

All experiments were conducted using 10-fold cross validation. To create the folds, duplicate records were grouped together, and the resulting clusters were randomly assigned to the folds. This method was chosen because randomly assigning records to different folds would dramatically reduce the number of duplicate records that can be potentially identified in each fold, as well as corrupt training data since some of the duplicate pairs would be split between the training and testing folds.

In each trial, the training phase of MARLIN was performed using nine training partitions, and then tested on the remaining test partition. Because the sizes of our datasets allowed computing distances between all

[1] http://cora.whizbang.com

Table 1: Sample duplicate records from the CORA database

authors	title	venue	address	year	pages
Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby	Information, prediction, and query by committee	Advances in Neural Information Processing System	San Mateo, CA	1993	pages 483-490
Freund, Y., Seung, H. S., Shamir, E., & Tishby, N.	Information, prediction, and query by committee	Advances in Neural Information Processing Systems	San Mateo, CA.	–	(pp. 483-490).

Table 2: Sample duplicate records from the RESTAURANT database

name	address	city	phone	cuisine
Second Avenue Deli	156 2nd Ave. at 10th St.	New York	212/677-0606	Delicatessen
Second Avenue Deli	156 Second Ave.	New York City	212-677-0606	Delis

Table 3: Sample duplicate records from the MAILING database

first	last	street address	city
Tsy C	Dodgson	18 Lilammal Ave 3k1	Christina MT 59423
Tessy	Dodgeson	PO Box 3879	Christina MT 59428

pairs of records, we did not employ the sorted neighborhood or canopies approaches to limit the number of potential duplicates. Either of the approaches, however, could be used for evaluating accuracy of duplicate detection on larger datasets.

After computing distances between all pairs of potential duplicates, the pair of records with the highest similarity was labeled as a duplicate, and the transitive closure of groups of duplicates was updated. Precision, recall and F-measure defined over pairs of duplicates were computed after each iteration:

$$Precision = \frac{\#ofCorrectlyIdentifiedDuplicatePairs}{\#ofIdentifiedDuplicatePairs}$$

$$Recall = \frac{\#ofCorrectlyIdentifiedDuplicatePairs}{\#ofTrueDuplicatePairs}$$

$$F{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

As more pairs with lower similarity are labeled as duplicates, recall increases, while precision begins to decrease because the number of non-duplicate pairs erroneously labeled as duplicates increases. Precision was interpolated at 20 standard recall levels following the traditional procedure in information retrieval [1] (Fig.11 shows results for two additional recall levels of 0.925 and 0.975). Some of the graphs show only those portions of the curves that exhibit differences between approaches; precision results for recall values that are not shown on the graphs were identical for all curves.

Table 4: Maximum F-measure for detecting duplicate field values

Distance metric	CORA title	RESTAURANT name	RESTAURANT address	MAILING name	MAILING address
Levenshtein	0.870	0.843	0.950	0.867	0.878
Learned Levenshtein	0.902	**0.886**	0.975	**0.899**	0.897
Affine	0.917	0.883	0.870	0.923	0.886
Learned Affine	**0.971**	**0.967**	**0.929**	**0.959**	0.892

4.3 Results

4.3.1 Detecting duplicate field values

To evaluate the usefulness of adapting character-based distance metrics to a specific domain, we compared learned similarity metrics with their fixed-cost equivalents for the task of identifying equivalent field values. Because duplicate records may contain field values that are not equivalent, while non-duplicate records may contain equivalent entries in some of the fields, it would be erroneous to label all fields from equivalent records as duplicates. For example, if two different restaurant records appear in a database, one containing *"New York City"* in the `city` field, and another containing *"New York"*, it would be erroneous to consider the pair *("New York City", "New York")* a non-duplicate. To avoid this problem, we have manually relabeled the duplicates for some of the fields to evaluate the utility of different metrics in detecting duplicates for individual fields. We chose the most meaningful fields from the three datasets for these experiments: CORA `paper title` field, RESTAURANT `name` and `address` fields, and MAILING `street address` and `name` fields (the latter is a concatenation of `first name` and `last name` fields).

We have compared four distance metrics:

- Levenshtein edit distance [22], calculated as the minimum number of character deletions, insertions and substitutions of unit cost;
- Learned Levenshtein edit distance based on a generative model and trained using the Expectation-Maximi-zation procedure described in [20];
- String distance with affine gaps [9] using a substitution cost of 3, gap opening cost of 3, and gap extension cost of 1, which are commonly used parameters;
- Learned string distance with affine gaps described in Section 2.2, trained using Expectation-Maximization procedure in Fig.4 with edit operation probabilities smoothed at $\lambda = 10^{-12}$ and converted to the additive cost model as described in Section 2.3.

Results for field-level duplicate detection experiments are summarized in Table 4. Each entry in table contains the average of maximum F-measure values over 10 folds. Results for experiments where the difference between the learned and corresponding unlearned metric is significant at the 0.05 level using a 1-tailed t-test are presented in bold font. Figures 8-10 contain recall-precision curves for the performance of MARLIN on the CORA `paper title` field, the RESTAURANT `address` field, and the MAILING `name` field (which is a concatenation of `first name` and `last name` fields).

Performance improvements achieved when learned distance metrics were used instead of fixed-cost distance metrics for detecting field duplicates demonstrate that learnable distance metrics are able to approximate the relative importance of differences between strings for a specific field. This can be seen from the fact that precision-recall curves for learned distance metrics are above those for corresponding fixed-cost metrics on Figures 8-10, as well as from higher maximum F-measure values in Table 4. Results of all experiments except for the `address` field of the MAILING database demonstrate that taking gaps into account when constructing string alignments results in better estimates of string similarity for the task of

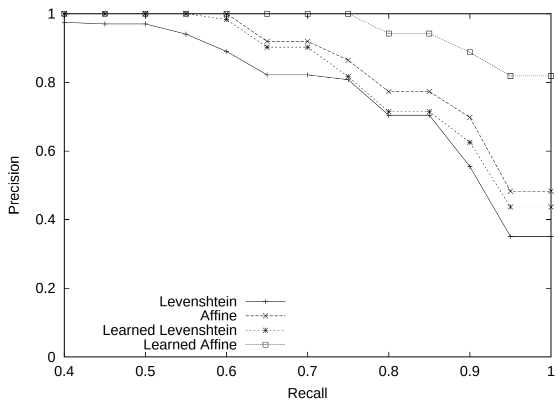

Figure 8: `Title` duplicate field-value detection results for the CORA dataset

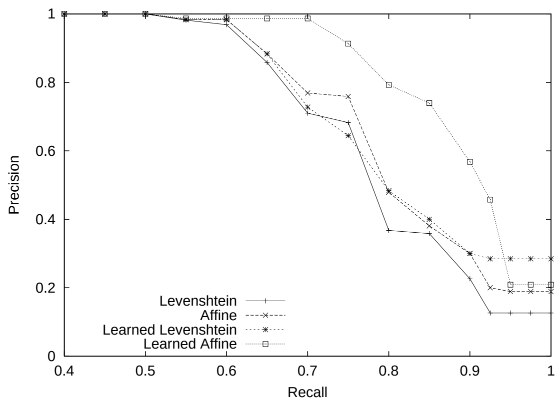

Figure 9: `Address` duplicate field-value detection results for the RESTAURANT dataset

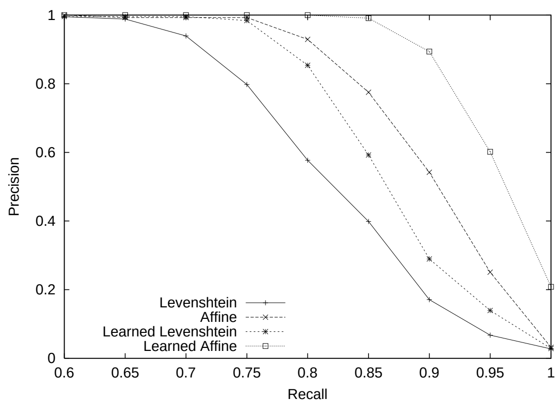

Figure 10: `Name` duplicate field-value detection results for the MAILING dataset

Table 5: Maximum F-measure for duplicate detection based on multiple fields

Classifier	Distance metric	CORA	RESTAURANT	MAILING
None (sum)	Affine	0.561	0.847	0.9431
None (sum)	Learned Affine	0.564	0.832	**0.991**
SVM	Affine	0.959	0.861	0.992
SVM	Learned Affine	0.958	**0.971**	**0.996**
SVM	Jaccard	0.983	0.971	0.961

detecting approximate duplicate field values. The fact that the results of all metrics were not significantly different on that field can be explained by the fact that a certain fraction of entries was heavily corrupted by substituting PO Box addresses, which are effectively impossible to match against the corresponding street address without using other fields such as `name` and `city`.

4.3.2 Record-level duplicate detection

Next, we evaluated the performance of MARLIN for multi-field (record-level) duplicate detection. The SVM implementation from the WEKA toolkit [26] that utilizes the sequential minimal optimization (SMO) algorithm [18] was used as the binary classifier. We have compared classifier-based similarity estimation to using the sum of distances from different fields as a non-trained record-level similarity measure. In these trials, distance values for individual fields were scaled by the largest string length that occurs in each field to correct for variation in field sizes. Either simple affine gap distance or learned string distance with affine gaps described above were used for computing similarity between values of each record field, corresponding to results on Figures 11-13 labeled as "Static" and "Adaptive". Classifier-based experiments are marked as "SVM", while experiments that used a sum of distances across fields are labeled as "Sum" on the figures. We also conducted additional experiments using the SVM for record-level classification based on Jaccard similarity as the distance metric for individual fields. Jaccard similarity was computed as shown in Eq.(3), where strings were split into tokens of alphanumeric characters.

Certain fields were not used in these experiments because they either provided too much information and resulted in near-100% precision and recall for all methods, e.g. `phone` in the RESTAURANT database, or they mostly contained missing field values, e.g. `editor` in CORA. Results for all experiments are summarized in Table 5. Again, results in bold font correspond to those experiments in which differences between using the learned and unlearned string metrics are significant at the 0.05 level using a 1-tailed t-test. All differences between the SVM and Sum approaches are significant at the 0.05 level using a 1-tailed t-test, except for the experiments that use unlearned string distance with affine gaps for the RESTAURANT dataset, and those that use learned string distance with affine gaps for the MAILING dataset.

From the results on the RESTAURANT and MAILING datasets we can conclude that using adaptive string distance metrics to compute similarity between field values makes a positive contribution when similarities from multiple fields are combined either in a simplistic manner by adding them, or by using them as record-pair attributes for classification. This means that better estimates of individual field similarities result in a more accurate calculation of the overall record similarity.

The fact that using learned distance metrics for estimating similarity between the fields did not aid the record-level matching process for the CORA dataset can be explained by the fact that most duplicates in this dataset have either very minor differences (such as abbreviations of authors' names), or drastic differences such as misplaced slots (e.g. authors' name in the `title` field), or missing features, such as `year` or `pages`. The sporadic and highly varying nature of these differences prevented trained string distance from capturing them. These domain peculiarities also explain the good performance of duplicate detection using

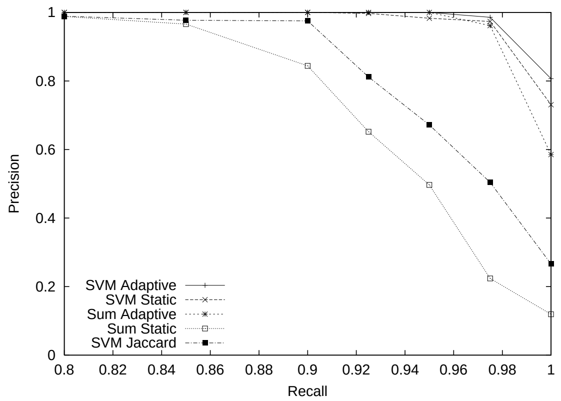

Figure 11: Duplicate detection results for MAILING dataset based on `first name`, `last name`, `street address` and `city` fields

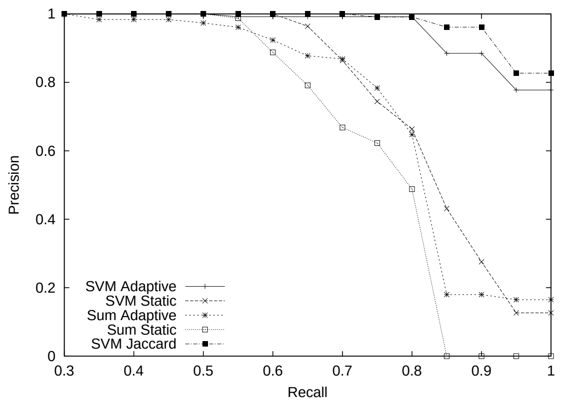

Figure 12: Duplicate detection results for RESTAURANT dataset based on `name`, `address`, `city` and `cuisine` fields

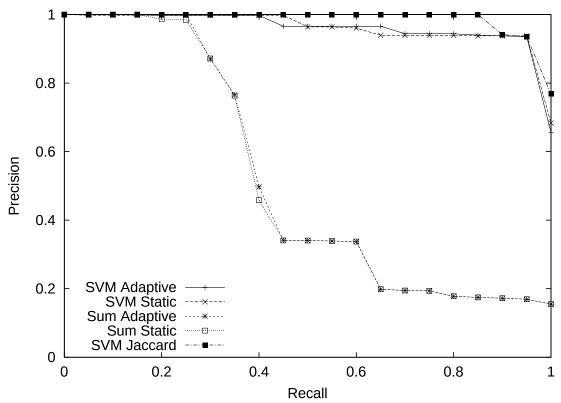

Figure 13: Duplicate detection results for CORA dataset based on `author`, `title`, `venue`, `year` and `pages` fields

Jaccard similarity to compare field values.

Limitations of using Jaccard similarity are highlighted by the results on the MAILING dataset. Because many duplicates are corrupted by typos, token-based distance metrics are not able to capture the degree of similarity between strings with minor variations in several characters. This result demonstrates that character-based metrics are particularly useful for detecting duplicates among shorter strings with minor variations, such as those resulting from OCR errors for scanned data or from typographic errors.

We also ran trials which combined character-based metrics (static and adaptive string distance with affine gaps) and token-based metrics (Jaccard similarity). These experiments resulted in near-100% precision and recall, without significant differences between static and adaptive field-level metrics. Similar results were obtained when common prefix and common suffix lengths were used as field-level distance metrics along with the character-based metrics used above. This demonstrates that combining character- and token-based distance metrics, such as learned string distance with affine gaps and Jaccard similarity, is clearly an advantage of the two-level approach implemented in MARLIN. Current datasets did not allow us to show the benefits of adaptive metrics over their static prototypes in this scenario, but our initial results suggest that this can be demonstrated on more challenging datasets.

5 Related Work

The problem of identifying duplicate records in databases was originally described by Newcombe [17] as record linkage in the context of identifying medical records of the same individual from different time periods. Fellegi and Sunter [7] developed a formal mathematical problem description for record linkage and offered statistical methods for estimating matching parameters and error rates. In more recent work in statistics, Winkler proposed using EM-based methods for estimating error rates and optimal matching rules [24]. This work studied the duplicate detection problem for the specialized domain of census records, therefore all similarity metrics were hand-tuned for optimal performance in this domain.

Hernández and Stolfo [10] developed the sorted neighborhood method for limiting the number of potential duplicate pairs that require distance computation, while McCallum et. al. proposed the canopies clustering algorithm [12] for the task of matching scientific citations. Monge and Elkan developed the iterative merging algorithm based on the union-find data structure [14] and showed the advantages of using a string distance metric that allows gaps [13]. Cohen et. al. described the problem of duplicate detection as database hardening: inferring the most likely underlying databases without duplicates (a "hard" database) given a database containing duplicates (a "soft" database) [3]. They proved NP-hardness of solving the problem optimally and proposed a nearly linear time algorithm for finding a local optimum using the union-find data structure.

In all of these approaches fixed-cost similarity metrics were used to compare database records. The only previous work on adaptive duplicate detection that we know of is the approach described by Cohen in [2], which learns how to combine multiple similarity metrics to identify duplicates, but does not adaptively tune the underlying field-similarity metrics themselves.

6 Future Work

Extending the metric learning approach to token-based distance metrics is a promising avenue for research. Because in some databases differences between duplicate records may take the form of commonly added and deleted tokens, it would be desirable to develop learning methods for token-based metrics, such as Jaccard similarity or vector-space cosine distance. Previous work on semi-supervised clustering [4] has shown the usefulness of a similar approach: learning weights of individual words when calculating distance between documents using Kullback-Leibler divergence.

Another area for future work lies in generalizing edit distance to include macro-operators for inserting and deleting common substrings, e.g. deleting "Street" in address fields. Certain common abbreviations and unimportant substrings account for many differences between records, and detecting these should improve our results. The string distance model with gaps would be particularly useful for this task, since it would allow discovering useful deletion sequences by counting the frequencies of common gaps. Substructure discovery methods [5] could also be used to identify useful edit operation sequences that include different edit operations.

7 Conclusions

Duplicate detection is an important problem in "data cleaning," and an adaptive approach that learns to identify duplicate records for a specific domain has clear advantages over a static, domain-independent method. Our approach uses learning at two levels. First, similarity metrics are trained to identify duplicate values for each field. Second, multiple similarity metrics for each field are combined to learn a final function for identifying duplicate records. Experimental results demonstrate that this approach detects duplicates more accurately than competing static approaches. In addition, results demonstrate that both levels of adaptation independently contribute to improving the overall accuracy of the system.

8 Acknowledgments

We would like to thank William Cohen for providing us the CORA dataset, Nick Kushmerick for useful discussion and Sheila Tejada's RESTAURANT dataset, and Mauricio Hernández for providing us the mailing address database generator used to create the MAILING dataset.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, 1999.

[2] W. Cohen and J. Richman. Learning to match and cluster entity names. In *ACM SIGIR-2001 Workshop on Mathematical/Formal Methods in Information Retrieval*, New Orleans, LA, Sept. 2001.

[3] W. W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 255–259, Boston, MA, Aug. 2000.

[4] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. Unpublished manuscript. Available at `http://www-2.cs.cmu.edu/~mccallum/`.

[5] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[6] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[7] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[8] R. Ghani, R. Jones, D. Mladenić, K. Nigam, and S. Slattery. Data mining on symbolic knowledge extracted from the web. In D. Mladenić, editor, *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, pages 29–36, Boston, MA, Aug. 2000.

[9] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, New York, 1997.

[10] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD-95)*, pages 127–138, San Jose, CA, May 1995.

[11] V. I. Levenshtein. Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, Feb. 1966.

[12] A. K. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pages 169–178, Boston, MA, Aug. 2000.

[13] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 267–270, Portland, OR, Aug. 1996.

[14] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, May 1997.

[15] U. Y. Nahm and R. J. Mooney. Using information extraction to aid the discovery of prediction rules from texts. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, pages 51–58, Boston, MA, Aug. 2000.

[16] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[17] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, 1959.

[18] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, Cambridge, MA, 1999.

[19] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[20] E. S. Ristad and P. N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5), 1998.

[21] G. Salton. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[22] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[23] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, 1995.

[24] W. E. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, U.S. Bureau of the Census, Wachington, DC, 1994.

[25] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, Wachington, DC, 1999.

[26] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.