Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery
and Data Mining(KDD-2003), Washington DC, pp.39-48, August, 2003

Adaptive Duplicate Detection Using Learnable String Similarity Measures

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

ABSTRACT

The problem of identifying approximately duplicate records in databases is an essential step for data cleaning and data integration processes. Most existing approaches have relied on generic or manually tuned distance metrics for estimating the similarity of potential duplicates. In this paper, we present a framework for improving duplicate detection using trainable measures of textual similarity. We propose to employ learnable text distance functions for each database field, and show that such measures are capable of adapting to the specific notion of similarity that is appropriate for the field's domain. We present two learnable text similarity measures suitable for this task: an extended variant of learnable string edit distance, and a novel vector-space based measure that employs a Support Vector Machine (SVM) for training. Experimental results on a range of datasets show that our framework can improve duplicate detection accuracy over traditional techniques.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Data cleaning, record linkage, distance metric learning, trained similarity measures, string edit distance, SVM applications

1. INTRODUCTION

Databases frequently contain field-values and records that refer to the same entity but are not syntactically identical. Variations in representation can arise from typographical errors, misspellings, abbreviations, as well as integration of multiple data sources. Variations are particularly pronounced in data that is automatically extracted from unstructured or semi-structured documents or web pages [16, 3]. Such approximate duplicates can have many deleterious effects, including preventing data-mining algorithms from discovering important regularities. This problem is 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 was referred to as record linkage [18, 25], the merge/purge problem [10], duplicate detection [15, 22], hardening soft databases [3], reference matching [13], and entity-name clustering and matching [4]. Typically, standard string similarity metrics such as edit distance [9] or vector-space cosine similarity [1] are used to determine whether two values or records are alike enough to be duplicates. Some more recent work [4, 22, 23] has investigated the use of pairing functions that combine multiple standard metrics.

Because an estimate of similarity between strings can vary significantly depending on the domain and specific field under consideration, traditional similarity measures may fail to estimate string similarity correctly. At the token level, certain words can be informative when comparing two strings for equivalence, while others are ignorable. For example, ignoring the substring "Street" may be acceptable when comparing addresses, but not when comparing names of people (e.g. "Nick Street") or newspapers (e.g. "Wall Street Journal"). At the character level, certain characters can be consistently replaced by others or omitted when syntactic variations are due to systematic typographical or OCR errors. Thus, accurate similarity computations require adapting string similarity metrics for each field of the database with respect to the particular data domain.

Rather than hand-tuning a distance metric for each field, we propose to use trainable similarity measures that can be learned from small corpora of labeled examples, and thus adapt to different domains. We present two such string similarity measures. The first one utilizes the Expectation-Maximization (EM) algorithm for estimating the parameters of a generative model based on string edit distance with affine gaps. The other string similarity measure employs a Support Vector Machine (SVM) [24] to obtain a similarity estimate based on the vector-space model of text. The character-based distance is best suited for shorter strings with minor variations, while the measure based on vector-space representation is more appropriate for fields that contain longer strings with more global variations.

Our overall system, MARLIN (Multiply Adaptive Record Linkage with INduction), employs a two-level learning approach. First, string similarity measures are trained for every database field so that they can provide accurate estimates of string distance between values for that field. Next, a final predicate for detecting duplicate records is learned from similarity metrics applied to each of the individual fields. We again utilize Support Vector Machines for this task, and show that they outperform decision trees, the classifier used in prior work [23, 22]. We evaluate our approach on several real-world data sets containing duplicate records and show that MARLIN can lead to improved duplicate detection accuracy over traditional techniques.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGKDD '03, August 24-27, 2003, Washington, DC, USA
Copyright 2003 ACM 1-58113-737-0/03/0008 ...$5.00.

2. LEARNABLE STRING DISTANCE

2.1 Background

Most traditional methods for calculating string similarity can be roughly separated into two groups: character-based techniques and vector-space based techniques. The former rely on character edit operations, such as deletions, insertions, substitutions and subsequence comparison, while the latter transform strings into vector representation on which similarity computations are conducted.

The best-known character-based string similarity metric is Levenshtein distance, defined as the minimum number of insertions, deletions or substitutions necessary to transform one string into another. Needleman and Wunsch [17] extended the 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 between two strings of lengths l_1 and l_2 can be computed using a dynamic programming algorithm that constructs three matrices in $O(l_1 l_2)$ computational time . The following recurrences are used to construct the matrices, where $c(x_i, y_j)$ denotes the cost of the edit operation that aligns i-th character of string x to j-th character of string y, while $c(x_i, \epsilon)$ and $c(\epsilon, y_j)$ are insertion and deletion costs for the respective characters. Matrix M represents a minimum-cost string alignment that ends with matched characters, while matrices I and D represent alignments that end with a gap in one of the two strings [9]:

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

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

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

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

While character-based metrics work well for estimating distance between strings that differ due to typographical errors or abbreviations, they become computationally expensive and less accurate for larger strings. When differences between equivalent strings are expressed by multiple words that are added, deleted or transposed, character-based metrics may assign high cost to non-matching string segments, producing low similarity scores for strings with a few word-level differences. The vector-space model of text [1] avoids this problem by viewing strings as "bags of tokens" and disregarding the order in which the tokens occur in the strings. Given a vocabulary of n possible tokens in a corpus, strings are represented as sparse n-dimensional vectors of real numbers, where every non-zero component corresponds to a token present in a string. TF-IDF is the most popular method for computing the weights of the vector representation; it takes into account token frequencies within a particular string and over the entire corpus. Similarity between two strings x and y is then computed as normalized dot product between their vector-space representations $\mathbf{x}$ and $\mathbf{y}$ (cosine of the angle between them):

$$Sim(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x} \cdot \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{\sum_{i=1}^{d} x_i y_i}{\|x\| \|y\|} \quad (2)$$

Because vectors representing the strings are highly sparse, vector-space cosine similarity is very efficient when implemented properly, and provides a reasonable off-the-shelf metric for larger strings and text documents.[1]

2.2 Learnable Distance Metrics

2.2.1 Learnable String Edit Distance with Affine Gaps

Different edit operations have varying significance in different domains. For example, a digit substitution makes a major difference in a street address since it effectively changes the house number, while a single letter substitution is semantically insignificant because it is more likely to be caused by a typo or an abbreviation. Therefore, adapting string edit distance to a particular domain requires assigning different weights to different edit operations.

In prior work, Ristad and Yianilos [21] have developed a generative model for Levenshtein distance along with an Expectation-Maximization algorithm that learns model parameters using a training set consisting of matched strings. We propose a similar stochastic model for the edit distance with affine gaps. The distance between two strings corresponds to a particular *alignment* of the strings' characters, which may include non-matching regions. An alignment can be viewed as a sequence of pairs $\langle x_i, y_j \rangle, \langle x_i, \epsilon \rangle, \langle \epsilon, y_j \rangle$, where x_i and y_j are the corresponding strings' characters, and ϵ is indicates a gap. Each character pair then represents an edit operation: a substitution or an insertion/deletion denoting a gap (matched characters can be thought of as a special case of substitution). The overall alignment represents a sequence of edit operations that transform one string into another with an associated cost.

The affine-gap alignment can be modeled by a simple stochastic transducer shown in Fig.1 below. The three matrices of the original model (1) correspond to accumulated forward probabilities for an alignment that ends in one of the three states. A particular alignment of two strings is generated by this model as a sequence of traversals along the edges. Each traversal is accompanied by an emission of a character pair sampled from a probability distribution of the state that is reached via each traversal.

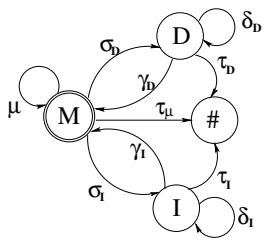

Figure 1: Generative model for string distance with affine gaps

[1]While string edit distance with affine gaps obeys the triangle inequality [9], and is therefore a metric in the strict mathematical sense, this is not true for normalized dot product. However, because the process of duplicate detection only requires a measure that accurately *ranks* pairs of strings with respect to their relative similarities, measures that do not satisfy the triangle inequality are satisfactory. Therefore, we will not restrict our discussion to measures that obey the triangle inequality and will use the term "metric" as a synonym of "measure".

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.

The production starts in state M and terminates when the special state # is reached. Transitions σ_D and σ_I from the matching state M to either the deletion state D or the insertion state I correspond to a gap in the alignment of the strings. A gap is ended when the edge γ_D (or γ_I) is traversed back to the matching state. Remaining in state M by taking edge μ corresponds to a sequence of substitutions or exact matches, while remaining in states I and D is analogous to extending a gap in either the first or the second string. The sum of transition probabilities must be normalized in each state for the model to be complete.

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 $\langle$/, -$\rangle$ for phone numbers, or deletion $\langle$., $\epsilon\rangle$ for addresses.

This generative model is similar to one given for amino-acid sequences in [6] with two important 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 that are outgoing from the same state.

Given two strings, x^T of length T and y^V of length V, probabilities of generating the pair of prefixes $(x^T_{1\ldots t}, y^V_{1\ldots v})$ and suffixes $(x^T_{t+1\ldots T}, y^V_{v+1\ldots V})$ can be computed using dynamic programming in standard forward and backward algorithms in $O(TV)$ time [20].

Then, 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 a variant of the Baum-Welch algorithm, shown in Fig.2, which is an Expectation-Maximization procedure for learning parameters of generative models [20]. The training procedure iterates between two steps, where in the first 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 character pair emission. In the MAXIMIZATION procedure all model parameters are updated using the collected expectations. Pseudo-code for the algorithms can be found in [2].

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: $d(x^T, y^V) = -\log p(x^T, y^V)$, and then obtaining the posterior probability. Numerical underflow may occur when these computation are performed for long strings; this problem can be resolved by mapping all computations into logarithmic space or by periodic scaling of all values in matrices M, D and I [21].

2.2.2 Practical Considerations

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 $a \in A$; $\tau = \tau_I = \tau_D$; $\gamma = \gamma_I = \gamma_D$; $\delta = \delta_I = \delta_D$; $\sigma = \sigma_I = \sigma_D$.

Input: A set of equivalent strings $\mathcal{S} = \{(x^{T_i}, y^{V_i}),\ x^{T_i} \approx y^{V_i}\}$
Output: A set of parameters for edit distance with affine gaps that minimizes distance for each $(x, y) \in \mathcal{S}$
Method:
until convergence
  for each $(x^{T_i}, y^{V_i}) \in \mathcal{S}$
    EXPECTATION-STEP: Use forward and backward algorithms to accumulate the expected number of occurrences $E[\langle x_j, y_k\rangle]$ for each edit operation used to align x^{T_i} and y^{V_i}, as well as $E[\mu], E[\sigma_I], E[\sigma_D], E[\delta_I], E[\delta_D], E[\gamma_I], E[\gamma_D]$.
  end
  MAXIMIZATION-STEP: Update all transition and emission probabilities using the expected numbers of occurrences and re-normalize.
end

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

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 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. Two steps are 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 reasonable minimum value λ. 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, the cost of producing matching pairs of characters is set to zero. The overall distance obtained from the model is normalized by the sum of string lengths to correct for the "increasing distance for longer exact duplicates" problem. The resulting metric can be viewed as a hybrid between the generative model and the original fixed-cost model.

2.2.3 Learnable Vector-space Similarity

The TF-IDF weighting scheme is useful for similarity computations because it attempts to give tokens weights that are proportional to the tokens' relative importance. However, the true contribution of each token to similarity is domain-specific. For example, suppose that duplicate detection is conducted on a database that contains street addresses. If there are several records of addresses from the same street, the street name token (for example, `'34th'`)

will have a lower weight due to its many occurrences across the corpus resulting in a lower IDF value. At the same time, some addresses may contain the token `'Square'`, which then will be assigned approximately the same weight as `'34th'` since it may be as common. While two addresses on the same street are more similar than two addresses that are both on squares, the generic TF-IDF cosine similarity is incapable of making the distinction.

If training data in the form of labeled pairs of duplicate and non-duplicate strings is available, it can be used to learn a similarity function that will give more weight to those tokens that are good indicators of actual similarity. Let us rewrite Eq.(2) in the following form:

$$Sim(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \frac{x_i y_i}{\|x\| \|y\|} \quad (3)$$

Every component $\frac{x_i y_i}{\|x\| \|y\|}$ of the sum corresponds to an i-th token in the vocabulary. This expression can be thought of as the i-th component of a d-dimensional vector that is being classified. Then the vector space similarity computation between two strings can be viewed as two consecutive steps:
(i) A d-dimensional *pair vector* $\mathbf{p}^{(x,y)} = \langle \frac{x_i y_i}{\|x\| \|y\|} \rangle$ is created, each component of which corresponds to the product of weights for the corresponding token in the vocabulary;
(ii) The summation is performed, which is equivalent to the pair instance $\mathbf{p}^{(x,y)}$ being classified by a perceptron with unit weights as either belonging to the equivalent-string class (output is 1), or different-string class (output is 0). The perceptron output value corresponds to the confidence of the prediction.

Provided training data in the form of equivalent-string pairs $\mathcal{S} = \{(x, y),\ x \approx y\}$ and different-string pairs $\mathcal{D} = \{(x, y),\ x \not\approx y\}$, it becomes possible to train a classifier for step (ii) to produce outputs that correspond to the desired categorization of string pairs. The trained classifier would be able to distinguish between features that are informative for similarity judgments, and those that are not, adapting the computation to the domain-specific notion of similarity that is contained in the training data.

A classifier appropriate for this task should be able to learn well from limited training sets of high-dimensional data. It also must produce meaningful confidence estimates that correspond to relative likelihood of belonging to equivalent-string or different-string classes for different string pairs. Another key requirement is that the the hypothesis learned by the classifier must be independent of the relative sizes of the positive and negative training sets, since the proportion of duplicate pairs in the training set is likely to be much higher than in the actual database where duplicates are detected.

Support vector machines [24] satisfy all of these requirements. SVMs classify input vectors $\mathbf{p}^{(x,y)}$ by implicitly mapping them via the "kernel trick" to a high-dimensional space where the two classes ($\mathcal{S}$, equivalent-string pairs, and $\mathcal{D}$, different-string pairs) are separated by a hyperplane. The output of the classifier is the distance between $\phi(\mathbf{p}^{(x,y)})$, the image of the pair vector in high-dimensional space, and the separating hyperplane. This distance provides an intuitive measure of similarity: those pair vectors that are classified with a large margin as belonging to $\mathcal{S}$ correspond to pairs of strings that are highly similar, while those classified with a large margin as belonging to $\mathcal{D}$ represent dissimilar pairs of strings.

The output of an SVM has the form $f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{p_i}, \mathbf{x}) + b$, where $K(\mathbf{x}, \mathbf{y})$ is a kernel function, and α_i is the Lagrangian coefficient corresponding to i-th training pair vector $\mathbf{p_i}$, obtained from the solution to the quadratic optimization problem. Because SVM training algorithms are independent of the dimensionality of the feature space (only the margin on the training data affects hypothesis choice), they are guaranteed not to overfit the solution to the training pairs.

Methods for obtaining calibrated posterior probabilities from the SVM output [19] could be used to obtain a probabilistically meaningful similarity function at this point. Because in the deduping framework we are only concerned with obtaining a correct *ranking* of pairs with respect to their true similarities, the distance from the hyperplane provides this without additional model fitting. The SVM output function $f(\mathbf{x})$ is bounded; the specific range of output values depends on the choice of kernel function K and the training set. For example, for radial basis function kernels $K(\mathbf{x}, \mathbf{x}_i) = \exp(-\frac{\|\mathbf{x}-\mathbf{x_i}\|^2}{2\sigma^2})$ (a very loose) bound is $|f(\mathbf{x})| \leq \sum_{i=1}^{l} \alpha_i + b$. Given upper and lower bounds $f_{\min} \leq f(\mathbf{x}) \leq f_{\max}$ for a particular kernel function and training set, it is trivial to obtain a similarity value in the $[0; 1]$ range from the SVM output:

$$Sim(s_1, s_2) = \frac{f(\mathbf{p}^{(s_1 s_2)}) - f_{\min}}{f_{\max} - f_{\min}} \quad (4)$$

The overall approach for obtaining a similarity value based on vector space using SVMs is shown on Fig.3, and the training algorithm is presented on Fig.4. When similarity between strings x and y needs to be computed, they are transformed to vector-space representations, and the pair vector $\mathbf{p}^{(x,y)}$ is formed from individual components of their individual vector representations. The pair vector is then classified by the trained SVM, and the final similarity value is obtained from the SVM prediction $f(\mathbf{p}^{(x,y)})$ using Eq.(4).

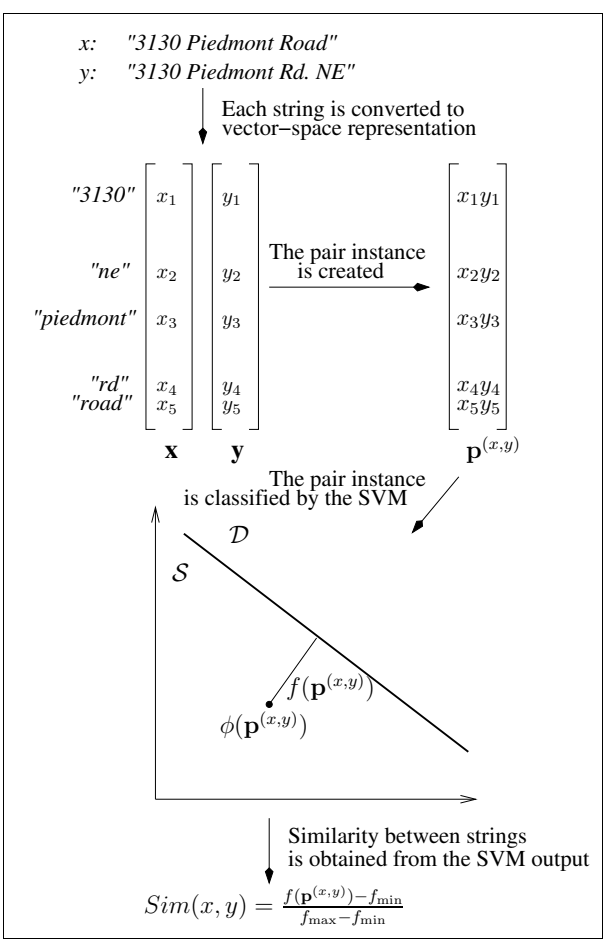

Figure 3: String similarity calculation for the SVM-based learnable vector-space model

Input: Sets of equivalent strings $\mathcal{S} = \{(x, y),\ x \approx y\}$ and non-equivalent strings $\mathcal{D} = \{(x, y),\ x \not\approx y\}$
Output: A string similarity function $Sim(x, y)$
Method:

1. Convert all strings in $\mathcal{S}$ and $\mathcal{D}$ to vector representation
2. Create positive and negative training sets $\mathcal{S}_p$ and $\mathcal{D}_p$ by constructing pair instances $\mathbf{p}^{(x,y)} = \langle \frac{x_i y_i}{\|x\|\|y\|} \rangle$: $\mathcal{S}_p = \{\mathbf{p}^{(x,y)},\ (x, y) \in \mathcal{S}\}$; $\mathcal{D}_p = \{\mathbf{p}^{(x,y)},\ (x, y) \in \mathcal{D}\}$
3. Train an SVM on $\mathcal{S}_p$ and $\mathcal{D}_p$ to obtain the classification function $f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{p_i}, \mathbf{x}) + b$, $f_{\min} \leq f(\mathbf{x}) \leq f_{\max}$
4. Return $Sim(x, y) = \frac{f(\mathbf{p}^{(x,y)}) - f_{\min}}{f_{\max} - f_{\min}}$

Figure 4: Training algorithm for SVM-based vector space similarity

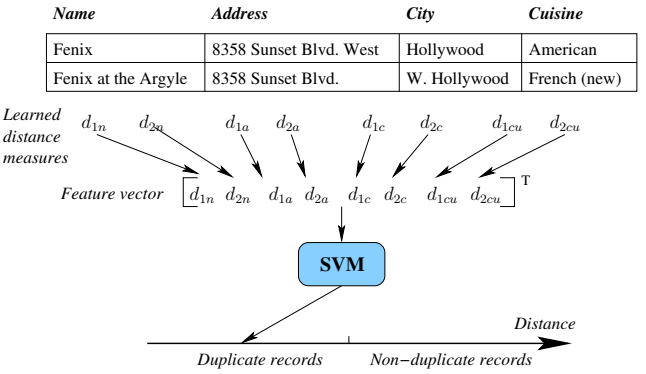

Figure 5: Computation of record similarity from individual field similarities

3. RECORD-LEVEL SIMILARITY

3.1 Combining similarity across multiple fields

When the distance between records composed of multiple fields is calculated, it is necessary to combine similarity estimates from individual fields in a meaningful manner. Because correspondence between overall record similarity and single-field similarity can vary greatly depending on how informative the fields are, it is necessary to weight fields according to their contribution to the true distance 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 that computes a "pairing function" [4]. Given a database that contains records composed of k different fields and a set $D = \{d_1(\cdot,\cdot), \ldots, d_m(\cdot,\cdot)\}$ of distance metrics, we can represent any pair of records by an mk-dimensional vector. Each component of the vector represents similarity between two field values of the records that is calculated using one of the m distance metrics.

Matched pairs of duplicate records can be used to construct a training set of such feature vectors by assigning them a positive class label. Pairs of records that are not labeled as duplicates implicitly form the complementary set of negative examples. If the transitive closure of matched pairs contains disjoint sets of duplicate records, this approach will result in noisy negative examples. Next, a binary classifier is trained using these training vectors to discriminate between pairs of records corresponding to duplicates and non-duplicates. Overall, this approach follows the same framework that is used for learnable vector-space string similarity in the previous section. Following the same reasoning, SVMs are a good classifier choice due to their resilience to noise and ability to handle correlated features well. The distance from the hyperplane provides a measure of confidence in the pair of records being a duplicate; it can be transformed to an actual similarity value using Eq.(4). Fig.5 illustrates this process of computing record similarity using multiple similarity measures over each field and a binary classifier to categorize the resulting feature vector as belonging to the class of duplicates or non-duplicates, resulting in a distance estimate. For each field of the database, two learnable distance measures, d_1 and d_2, are trained and used to compute similarity for that field. The values computed by these measures form the feature vector that is then classified by a support vector machine, producing a confidence value that represents similarity between the database records.

3.2 The overall duplicate detection framework

An overall view of our system, MARLIN, is presented in Fig.6.

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 and non-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. For example, if one record describing a restaurant contains `'Asian'` in the **Cuisine** field, and a duplicate record contains `'Seafood'`, a noisy training pair is formed that implies equivalence between these two descriptors. However, this issue does not pose a serious problem for our approach for two reasons. First, particularly noisy fields that are unhelpful for identifying record-level duplicates will be considered irrelevant by the classifier that combines similarities from different fields. Second, the presence of such pairs in the database indicates that there is a degree of similarity between such values, and using them in training allows the learnable metric to capture that likelihood.

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.

The duplicate detection phase starts with generation of potential duplicate pairs. Given a large database, producing all possible pairs of records and computing similarity between them is too expensive since it would require $O(n^2)$ distance computations. MARLIN utilizes the *canopies* clustering method [13] using Jaccard similarity, a computationally inexpensive metric based on an inverted index, to separate records into overlapping clusters ("canopies") of potential duplicates. Pairs of records that fall in each cluster then become candidates for a full similarity comparison shown in Fig.5.

Learned distance metrics are then 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 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 dupli-

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
fenix	8358 sunset blvd. west	hollywood	213/848-6677	american
fenix at the argyle	8358 sunset blvd.	w. hollywood	213-848-6677	french(new)

Table 3: Sample duplicate records from the *Reasoning* database

L. P. Kaelbling. An architecture for intelligent reactive systems. In Reasoning About Actions and Plans: Proceedings of the 1986 Workshop. Morgan Kaufmann, 1986
Kaelbling, L. P., 1987. An architecture for intelligent reactive systems. In M. P. Georgeff & A. L. Lansky, eds., Reasoning about Actions and Plans, Morgan Kaufmann, Los Altos, CA, 395 410

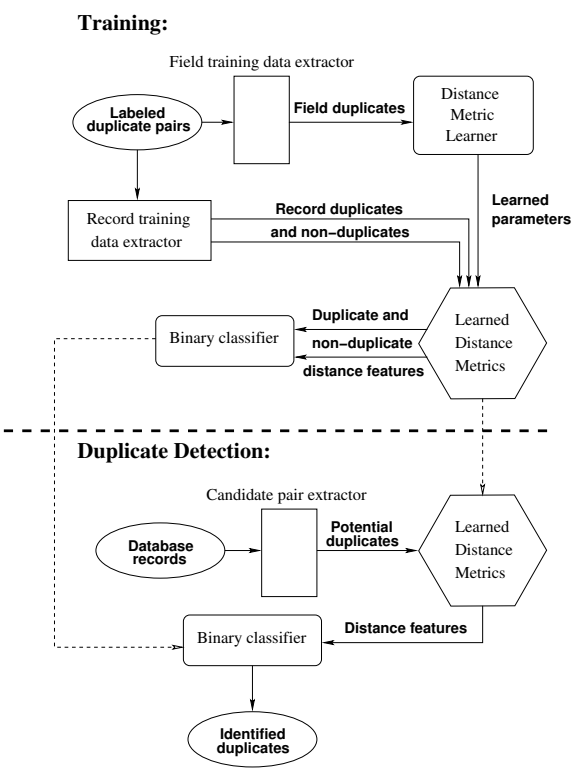

Figure 6: MARLIN overview

cates (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 possible 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 [15], MARLIN utilizes 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

Our experiments were conducted on six datasets. *Restaurant* is a database of 864 restaurant names and addresses containing 112 duplicates obtained by integrating records from Fodor's and Zagat's guidebooks. *Cora* is a collection of 1295 distinct citations to 122 Computer Science research papers from the Cora Computer Science research paper search engine. The citations were segmented into multiple fields such as **author**, **title**, **venue**, etc. by an information extraction system, resulting in some crossover noise between the fields. *Reasoning*, *Face*, *Reinforcement* and *Constraint* are single-field datasets containing unsegmented citations to computer science papers in corresponding areas from the *Citeseer* scientific literature digital library.[2] *Reasoning* contains 514 citation records that represent 196 unique papers, *Face* contains 349 citations to 242 papers, *Reinforcement* contains 406 citations to 148 papers, and *Constraint* contains 295 citations to 199 papers. Tables 1–3 contain sample duplicate records from the *Restaurant*, *Cora*, and *Reasoning* datasets.

4.2 Experimental Methodology

Every dataset was randomly split into 2 folds for cross-validation for each experimental run. A larger number of folds is impractical since it would result in few duplicate records per fold. To create the folds, duplicate records were grouped together, and the resulting clusters were randomly assigned to the folds, which resulted in uneven folds in some of the trials. All results are reported over 20 random splits, where for each split the two folds were used alternately for training and testing.

During each trial, duplicate detection was performed as described in Section 3.2. At each iteration, the pair of records with the highest similarity was labeled 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, where precision is the fraction of identified duplicate pairs that are correct, recall is the fraction of actual duplicate pairs that were identified, and F-measure is the harmonic mean of precision and recall:

[2] `http://citeseer.nj.nec.com/`

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

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

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

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].

4.3 Results

4.3.1 Detecting duplicate field values

To evaluate the usefulness of adapting string similarity measures to a specific domain, we compared learned distance metrics with their fixed-cost equivalents for the task of identifying equivalent field values. Along with the four single-field *Citeseer* datasets we chose two most meaningful fields from the *Restaurant* dataset - **name** and **address**.

We have compared four string similarity measures:

- Edit distance with affine gaps [9] using substitution cost of 2, gap opening cost of 3, gap extension cost of 1, and match cost of -1, which are commonly used parameters;
- Learned edit distance with affine gaps described in Section 2.2.1, trained using the EM algorithm shown in Fig.2 with edit operation probabilities smoothed at $\lambda = 10^{-5}$ and converted to the additive cost model as described in Section 2.2.2;
- Normalized dot product in vector space (cosine similarity), computed using TF-IDF weights after stemming and stopword removal;
- Learnable vector-space SVM-based similarity described in Section 2.2.3, implemented over TF-IDF representations after stemming and stopword removal. SVM implementation with the radial basis function kernel from the SVM^{light} package was used as the classifier [11] with $\sigma = 10$.

Results for field-level duplicate detection experiments are summarized in Table 4. Each entry in the table contains the average of maximum F-measure values over the 40 evaluated 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 paired t-test are presented in bold font. Figs. 7 and 8 contain recall-precision curves for the performance of MARLIN on the **name** and **address** fields of the *Restaurant* dataset respectively.

Relatively low precision of these two experiments is due to the fact that the duplicates on individual fields are very noisy: for example, several restaurants from different cities may have variations of the same name, and in these trials these variations would be considered a non-duplicate. However, results in the following section will show that a combination of individual field estimates provides an accurate approximation of overall record similarities. The comparison of results for learned metrics and the corresponding baselines shows that despite the noise, learned edit distance was able to adjust to the notion of similarity specific for each domain, while learned vector-space similarity improved over the standard vector-space similarity for half the domains.

It is peculiar that the learned vector space measure made more mistakes than unlearned cosine similarity for low recall values on

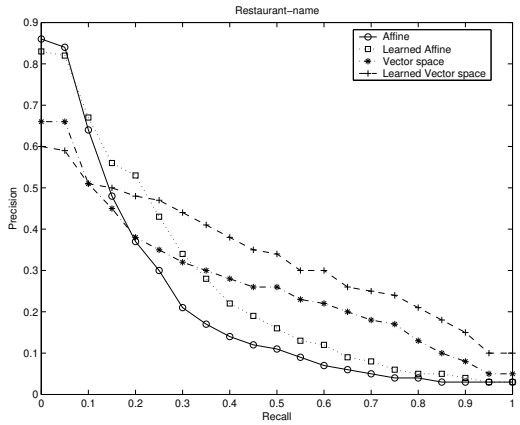

Figure 7: Field duplicate detection results for the name field of the *Restaurant* dataset

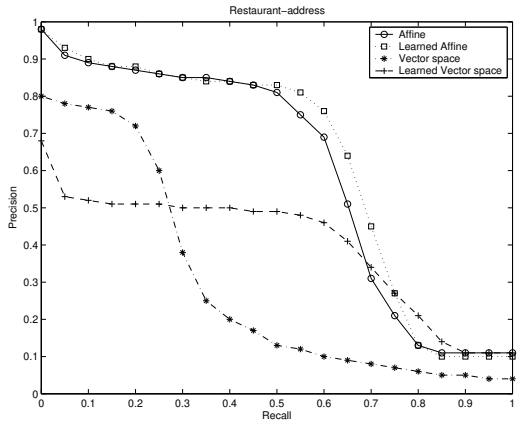

Figure 8: Field duplicate detection results for the address field of the *Restaurant* dataset

the **name** and **address** datasets, but has outperformed it for higher recall. We conjecture that cosine similarity was able to correctly identify the few duplicates that were transpositions of exact matches (e.g. `'Hotel Bel-Air'` and `'Bel-Air Hotel'`, but has failed to give sufficiently high similarity to more difficult duplicates (e.g. `'Art's Deli'` and `'Art's Delicatessen'`). The SVM-trained metric, on other hand, was able to generalize from the training examples, resulting in better similarity estimates across a range of more difficult duplicates while penalizing some of the "obvious" matches.

Overall, results of Table 4 show that learned affine edit distance outperforms both non-trained edit distance and vector-space cosine similarity for individual field duplicate detection. Visual inspection of the learned parameter values reveals that the parameters obtained by our training algorithm indeed capture certain domain properties that allow more accurate similarity computations. For example, for the **address** field of the *Restaurant* data, the lowest-cost edit operations after deleting a space are deleting 'e' and deleting 't' - which captures the fact that a common cause of street name duplicates are abbreviations from `''Street''` to `''Str''`.

4.3.2 Record-level duplicate detection

Next, we evaluated the performance of MARLIN for multi-field (record-level) duplicate detection. We again used the SVM^{light}

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

Distance metric	*Restaurant* **name**	*Restaurant* **address**	*Reasoning*	*Face*	*Reinforcement*	*Constraint*
Edit distance	0.290	0.686	0.927	0.952	0.893	0.924
Learned edit distance	**0.354**	**0.712**	**0.938**	**0.966**	**0.907**	**0.941**
Vector-space	0.365	0.380	0.897	**0.922**	**0.903**	0.923
Learned vector-space	**0.433**	**0.532**	**0.924**	0.875	0.808	0.913

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

Metric	*Restaurant*	*Cora*
Edit distance	0.904	0.793
Learned edit distance	**0.922**	**0.824**
Vector space	**0.919**	**0.867**
Learned Vector space	0.826	0.803

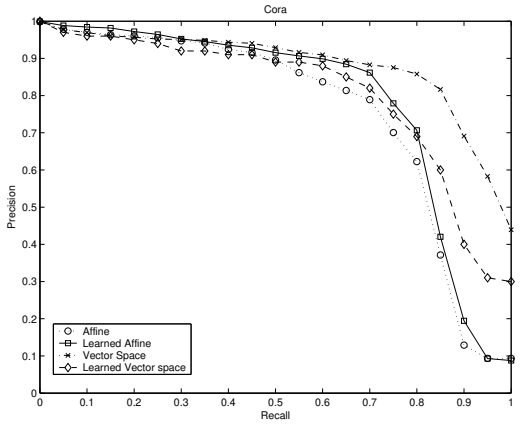

Figure 9: Duplicate detection results for the *Cora* dataset based on author, title, venue, year and pages fields

implementation of a support vector machine as the binary classifier that combines similarity estimates across the different fields to produce the overall measure of record similarity as shown on Fig.5.

We have compared the performance of learnable and baseline text similarity metrics for producing the similarity estimates of individual fields. Table 5 summarizes the results for the *Restaurant* and *Cora* datasets. 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 paired t-test. Figures 9 and 10 present the precision-recall curves for the experiments.

From these results we can conclude that using learnable string edit distance with affine gaps metrics to compute similarity between field values makes a positive contribution when similarities from multiple fields are combined. Thus, better estimates of individual field similarities result in a more accurate calculation of the overall record similarity.

Using the SVM-based vector-space learnable similarity did not lead to improvements over the original vector space cosine similarity; performance has in fact decreased. Given that results for field-based duplicate detection with learnable vector-space metric were mixed, this is not surprising. We conducted additional experiments where the folds were created not by assigning the equivalence classes of duplicate records to folds, but by randomly assigning individual instances. This resulted in duplicate pairs corresponding to the same object being in both training and test sets (such experiments were only possible for the *Cora* and *Citeseer*

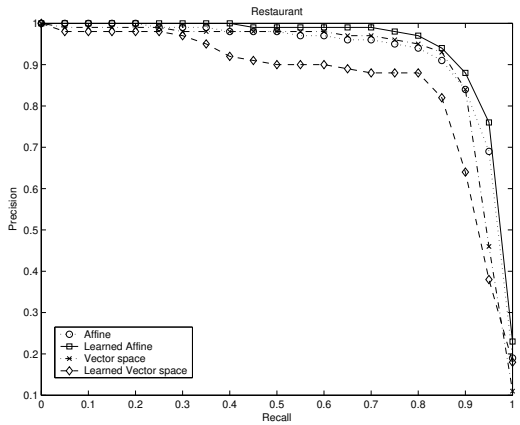

Figure 10: Duplicate detection results for the *Restaurant* dataset based on name, address, city and cuisine fields

datasets since there are at most 2 duplicates per equivalence class in the *Restaurant* dataset). In several of these experiments the learned vector-space metrics outperformed the unlearned baseline. This can be explained by the fact that the training algorithm used by SVM^{light} relies on the assumption that the training data comes from the same distribution as the test data. Separating equivalence classes into different folds results in different token distributions for the two sets, and the learned classifier is not suitable for producing accurate predictions on the test data as a result.

We also ran trials which combined character-based metrics (static and learnable string edit distance with affine gaps) and vector-space cosine similarity. These experiments resulted in near-100% precision and recall, without significant differences between static and adaptive field-level metrics. This demonstrates that combining character and token-based distance metrics, such as learned string distance with affine gaps and cosine similarity, is clearly an advantage of the two-level learning 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.

4.3.3 Comparison of classifiers for record-level duplicate detection

Previous work that employed classifiers to combine similarity estimates from multiple fields has utilized committees of decision tree learners [22, 23]. We compared performance of support vector machines to boosted decision trees for combining similarity estimates across the database fields to produce overall similarity of records. We conducted experiments for two settings: some using very limited training data (30 negative and 30 positive duplicate pair examples), and using large amounts of training data (using 500 randomly sampled negative pairs and up to 500 positive pairs - fewer were available for the *Restaurant* dataset due to the limited number of duplicates in it). The SVM^{light} implementation of a support vector machine with a radial basis function kernel was compared

with the WEKA package [26] implementation of alternating decision trees [8], a state-of-the-art algorithm that combines boosting and decision tree learning. Unlearned vector-space normalized dot product was used as the field-level similarity measure. Figs.11 and 12 illustrate the results on the *Restaurant* and *Cora* datasets.

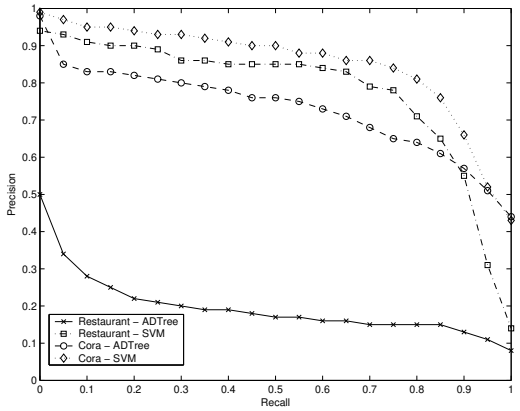

Figure 11: Duplicate detection results for *Restaurant* and *Cora* datasets using different record-level classifiers on limited training data

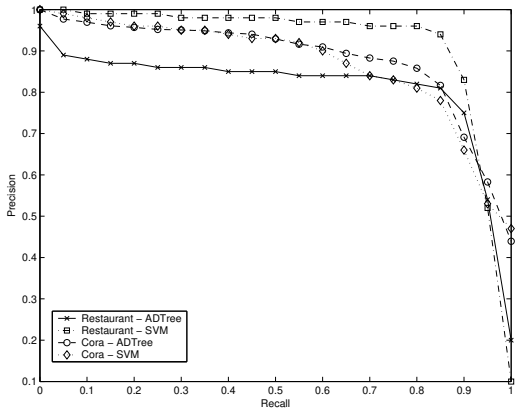

Figure 12: Duplicate detection results for *Restaurant* and *Cora* datasets using different record-level classifiers on large amounts of training data

These results show that support vector machines significantly outperform boosted decision trees when training data is limited, which is the most likely scenario for adaptive duplicate detection. While decision trees are reliable classifiers, obtaining calibrated confidence scores from them relies on probability estimates based on training data statistics over the tree nodes [27]. When little training data is available, such frequency-based estimates are very unreliable. As a result, the confidence of the decision tree classifier is an inaccurate measure of relative record similarity that leads to poor accuracy in the duplicate detection process.

5. RELATED WORK

The problem of identifying duplicate records in databases was originally identified by Newcombe [18] 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 theory 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 obtaining optimal matching rules [25]. That work was highly specialized for the domain of census records and used hand-tuned similarity measures.

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 [13] for the task of matching scientific citations. Monge and Elkan developed the iterative merging algorithm based on the union-find data structure [15] and showed the advantages of using a string distance metric that allows gaps [14]. Cohen et. al. [3] posed the duplicate detection task as an optimization problem, proved NP-hardness of solving the problem optimally, and proposed a nearly linear algorithm for finding a local optimum using the union-find data structure.

In recent work, Cohen and Richman have proposed an adaptive framework for duplicate detection that combines multiple similarity metrics [4]. Sarawagi and Bhamidipaty [22] and Tejada et. al. [23] developed systems that employ committee-based active learning methods for selecting record pairs that are informative for training the record-level classifier that combines similarity estimates from multiple fields across different metrics. In all of these approaches fixed-cost similarity metrics were used to compare the database records. We have shown that learnable similarity measures can be combined with trainable record-level similarity, and active learning techniques from prior work can be easily extended to include the distance measures that we proposed.

6. FUTURE WORK

We have proposed a general framework for using learnable string similarity measures in duplicate detection, and provided two algorithms for character-based and vector-space based text distances. There are several directions in which this approach can be extended.

The general classification-based framework for computing vector-space similarity can be improved by modifying the SVM training algorithm to avoid the overfitting issues that we encountered. The algorithm based on iterative decomposition of the quadratic optimization problem used by SVM^{light} needs to be extended to be robust to differences between distributions of test data and training data. This task is similar to transduction [24, 12] because it would require using unlabeled test data in the learning process, but with the fundamental departure from transduction in using unlabeled test data from a *different distribution*. Alternatively, the task of learning vector-space similarity between pairs of strings can be formalized as a parameter estimation or an optimization problem, and investigating statistical or mathematical programming methods that would incorporate regularization to deal with the distribution problem is a promising avenue for improvement.

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. The string distance model with gaps would be particularly useful for this task, since it would allow discovering useful deletion sequences by developing a stochastic model based on the gaps created when computing minimum-cost alignments. 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 static methods. Experimental results demonstrate that trainable similarity measures are capable of learning the specific notion of similarity that is appropriate for a specific domain. We presented two learnable distance measures that improve over character-based and vector-space based metrics and allow specializing them for specific datasets using labeled examples. We have also shown that support vector machines can be effectively utilized for some datasets both for string similarity and record similarity computations, outperforming traditional methods; we hope to improve on these initial results in our future work. Our overall framework for duplicate detection integrates previous work on adaptive methods with learnable similarity measures, leading to improved results.

8. ACKNOWLEDGMENTS

We would like to thank Steve Lawrence for providing us the *Citeseer* datasets, Sheila Tejada for the *Restaurant* dataset, and William Cohen for providing the *Cora* dataset. This research was supported by the National Science Foundation under grant IIS-0117308 and a Faculty Fellowship from IBM Corporation.

9. REFERENCES

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

[2] M. Bilenko and R. J. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX, Feb. 2002.

[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)*, Boston, MA, Aug. 2000.

[4] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta, 2002.

[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] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, 1999.

[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] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press, 1999.

[12] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, June 1999.

[13] 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.

[14] 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.

[15] 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.

[16] 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*, Boston, MA, Aug. 2000.

[17] 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.

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

[19] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 185–208. MIT Press, 1999.

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

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

[22] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta, 2002.

[23] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta, 2002.

[24] V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[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.

[27] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of 18th International Conference on Machine Learning (ICML-2001)*, Williamstown, MA, 2001.