Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web,
pp. 67-72, Acapulco, Mexico, August, 2003

Employing Trainable String Similarity Metrics for Information Integration

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 objects in databases is an essential step for the information integration process. 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 data field, and introduce an extended variant of learnable string edit distance based on an Expectation-Maximization (EM) training algorithm. Experimental results on a range of datasets show that this similarity metric is capable of adapting to the specific notions of similarity that are appropriate for different domains. Our overall system, MARLIN, utilizes support vector machines to combine multiple similarity metrics, which are shown to perform better than ensembles of decisions trees, which were employed for this task in previous work.

1 Introduction

When databases contain records that were collected from multiple information sources, they frequently include field-values and tuples that refer to the same entity, but are not syntactically identical. Variations in representation across sources can arise from differences in formats used to store data, typographical errors, and abbreviations. Variations are particularly pronounced in data that is automatically extracted from web pages and unstructured or semi-structured documents [Nahm and Mooney, 2000; Cohen *et al.*, 2000]. Such approximate duplicates can have many deleterious effects on other data integration tasks, including preventing data-mining algorithms from discovering important regularities. This problem is typically handled during a tedious manual de-duplication process.

Some previous work has addressed the problem of identifying duplicate records, where it was referred to as record linkage [Fellegi and Sunter, 1969; Winkler, 1999], the merge/purge problem [Hernández and Stolfo, 1995], duplicate detection [Monge and Elkan, 1997; Sarawagi and Bhamidipaty, 2002], hardening soft databases [Cohen *et al.*, 2000], reference matching [McCallum *et al.*, 2000], object identification [Tejada *et al.*, 2002], and entity-name clustering and matching [Cohen and Richman, 2002]. Typically, standard string similarity metrics such as edit distance [Gusfield, 1997] or vector-space cosine similarity [Baeza-Yates and Ribeiro-Neto, 1999] are used to determine whether two values or records are alike enough to be duplicates. Some more recent work [Cohen and Richman, 2002; Sarawagi and Bhamidipaty, 2002; Tejada *et al.*, 2002] 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. When syntactic variations are due to systematic typographical or OCR errors, certain characters can be consistently replaced by others or omitted. Certain abbreviations are also common to some domains, for example "Street" is frequently abbreviated to "Str" in addresses. Accurate similarity computations therefore 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, thus adapting to a specific domain. We present a trainable variant of edit distance with affine gaps, which is a widely used similarity metric for short and medium-length strings. Our metric is based on a three-state stochastic generative model, and an Expectation-Maximization (EM) algorithm is presented for estimating its parameters.

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 utilize support vector machines for this task, and show that they outperform boosted decision trees, the classifier used in prior work [Tejada *et al.*, 2002; Sarawagi and Bhamidipaty, 2002]. We evaluate our approach on several real-world data sets containing duplicate records and show that MARLIN improves duplicate detection accuracy over traditional techniques.

2 Learnable String Distance

2.1 Background

The most widely used similarity metric for short strings is Levenshtein distance, defined as the minimum number of insertions, deletions or substitutions necessary to transform one string into another. Its computation can be performed in time proportional to the strings' lengths using dynamic programming. Allowing contiguous sequences of mismatched characters, or gaps, in the alignment of two strings, lessens the penalty for insertion or deletion of complete substrings, and results in a better similarity estimate than Levenshtein distance, since it accomodates abbreviations and whole-word insertions and deletions.

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. Computing edit distance with affine gaps is performed via a dynamic programming algorithm that constructs three matrices that represent minimum-cost string alignments that end either matched characters, or with a gap in one of the two strings [Gusfield, 1997].

2.2 Learnable 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 [Ristad and Yianilos, 1998] 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 stochastic transducer shown in Fig.1 represents the generative model that produces *alignments* of characters between two strings that may include gaps. 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. An aligned pair of strings corresponds to a sequence of emissions of character pairs, or single characters from one of the strings that are aligned to a gap in the other string.

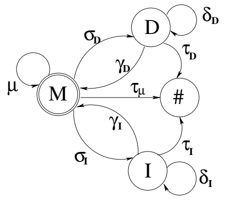

Figure 1: Generative model for string distance with affine gaps

Given an alphabet of symbols $A^* = A \bigcup \{\varepsilon\}$, 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 $E_i = \{\langle \varepsilon,a \rangle \mid a \in A\}$ and $E_d = \{\langle a,\varepsilon \rangle \mid a \in A\}$ are sets of insertion and deletion operations respectively. 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 $\sum_{e \in E_i} p(e) = 1$.

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.

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 [Rabiner, 1989].

Provided 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 [Rabiner, 1989]. 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. Complete pseudo-code for the algorithm can be found in [Bilenko and Mooney, 2002].

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

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 [Ristad and Yianilos, 1998].

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. To address this issue, the probability distribution over the set of edit operations, E, is smoothed by bounding each edit operation probability by some minimum value λ.

To improve efficiency, learned parameters of the generative distance model can be mapped to operation costs of the original edit distance with affine gaps by taking the negative logarithm of each probability. Distance calculation is then analogous to the classical three-matrix dynamic programming approach 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.

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. Availability of labeled duplicates allows using a binary classifier that computes a "pairing function" [Cohen and Richman, 2002]. 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 similrity between 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 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. A binary classifier is trained using these training vectors to discriminate between pairs of records corresponding to duplicates and non-duplicates.

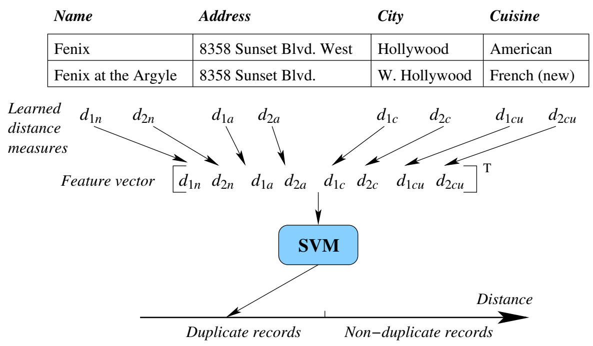

Figure 3: Computation of record similarity from individual field similarities

Support vector machines [Vapnik, 1998] are an appropriate classifier for this task due to their resilience to noise and ability to learn from few informative training instances. The distance from the separating hyperplane provides an appropriate measure of confidence in the pair of records being a duplicate. Fig.3 illustrates the process of computing record similarity using multiple distance metrics 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 the sample database of Fig.3, two learnable distance metrics, d_1 and d_2, are trained and subsequently used to compute similarity for each field. The values computed by these metrics 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

The training phase of our system, MARLIN, consists of two steps. First, 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. However, this issue does not pose a serious problem for our approach, since particularly noisy fields that are unhelpful for identifying record-level duplicates will be considered irrelevant by the classifier that combines similarities from different fields.

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.

In the duplicate detection phase, MARLIN utilizes the *canopies* clustering method [McCallum *et al.*, 2000] to avoid performing $O(n^2)$ distance computations between all pairs of database records. Using Jaccard similarity, a computationally inexpensive metric based on an inverted index, records are separated into overlapping clusters ("canopies") of potential duplicates. Pairs of records that fall in each cluster then become candidates for a full similarity comparison.

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 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. 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 a threshold that is 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 [Monge and Elkan, 1997], MARLIN utilizes the union-find data structure to store all database records in this

Table 1: 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)

step, which allows updating the transitive closure of identified duplicates incrementally in an efficient manner.

4 Experimental Evaluation

4.1 Datasets and Methodology

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 web sites. *Cora* is a collection of 1295 distinct citations to 122 Computer Science research papers from the Cora Computer Science research paper search engine. *Reasoning*, *Face*, *Reinforcement* and *Constraint* are single-field citation datasets from the *Citeseer* scientific literature digital library (`http://citeseer.nj.nec.com/`) *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. Table 1 contain sample duplicate records from the *Restaurant* dataset.

For each experimental run every dataset was randomly split into 2 folds for cross-validation by assigning all records for every unique underlying object to one of the folds. All results are reported over 20 random splits, where for each split the two folds were used alternately for training and testing.

During each run, 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 [Baeza-Yates and Ribeiro-Neto, 1999].

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 [Baeza-Yates and Ribeiro-Neto, 1999].

4.2 Results

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 [Gusfield, 1997] 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;

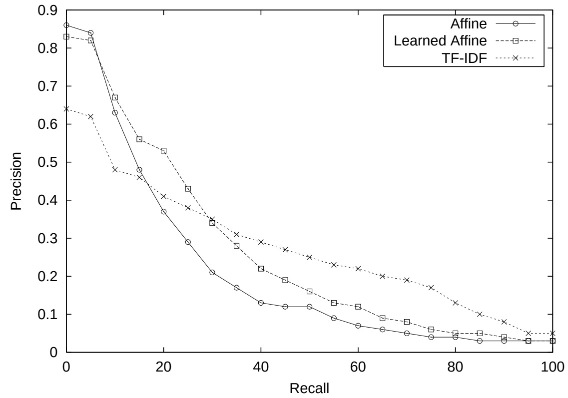

Figure 4: Field duplicate detection results for the Name field of the *Restaurant* dataset

- Learned edit distance with affine gaps described in Section 2.2, trained using the EM algorithm shown in Fig.2 with edit operation probabilities smoothed at $\lambda = 10^{-5}$;
- Normalized dot product in vector space (cosine similarity), computed using TF-IDF weights after stemming and stopword removal.

Results for field-level duplicate detection experiments are summarized in Table 2. 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 unlearned edit distance is significant at the 0.05 level using a 1-tailed t-test are presented in bold. Fig. 4 demonstrates the recall-precision curves for the performance of MARLIN on the **name** field of the *Restaurant* dataset.

Relatively low precision of the MARLIN field 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.

Overall, these results demonstrate that despite the noise learned affine edit distance is able to outperform 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 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''`.

Table 2: Maximum F-measure for detecting duplicate field values

Distance metric	*Restaurant* **name**	*Restaurant* **address**	*Reasoning*	*Face*	*Reinforcement*	*Constraint*
Edit distance	0.290	0.749	0.927	0.952	0.893	0.924
Learned edit distance	**0.354**	**0.787**	**0.938**	**0.966**	**0.907**	**0.941**
Vector-space	0.365	0.412	0.897	0.922	0.903	0.923

Table 3: Maximum F-measure for duplicate detection based on multiple fields

Metric	*Restaurant*	*Cora*
Edit distance	0.909	0.793
Learned edit distance	**0.928**	**0.824**
Vector space	0.917	0.867

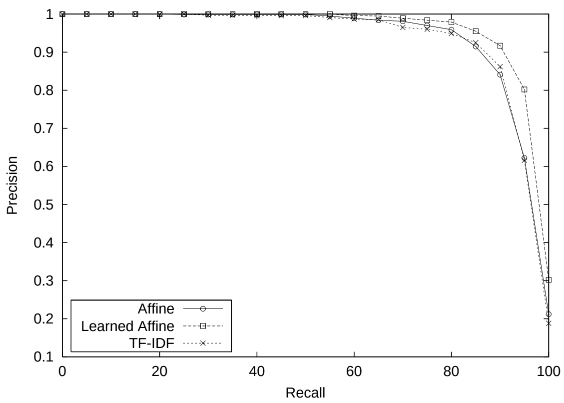

Figure 5: Duplicate detection results for the *Restaurant* dataset based on **name, address, city** and **cuisine** fields

Record-level duplicate detection

Next, we evaluated the performance of MARLIN for multi-field (record-level) duplicate detection. We used the SVM^{light} implementation of a support vector machine [Joachims, 1999] as the binary classifier that combines similarity estimates across the different fields to produce the overall measure of record similarity as shown on Fig.3.

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, and Fig.5 contains the recall-precision curve for the *Restaurant* dataset. The results demonstrate that using learnable string similarity metrics makes a positive contribution when similarities from multiple fields are combined.

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 is 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 for this scenario, but the initial results suggest that they can be demonstrated on more challenging datasets.

Comparison of classifiers for adaptive duplicate detection

Previous work that employed classifiers to combine similarity estimates from multiple fields has utilized committees of decision tree learners [Sarawagi and Bhamidipaty, 2002; Tejada *et al.*, 2002]. 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. Experiments were conducted for two scenarios: using very limited training data (30 negative and 30 positive duplicate pair examples), and using large amounts of training data (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 [Witten and Frank, 1999] implementation of alternating decision trees [Freund and Mason, 1999], a state-of-the-art algorithm that combines boosting and decision tree learning. Unlearned vector-space cosine similarity was used as the field-level similarity measure. Fig.6 illustrate the results on the *Restaurant* and *Cora* datasets for the the limited training data setting. For the large amount of training data, SVMs significantly outperformed decision trees on *Restaurant*, and performed slightly worse on *Cora* data.

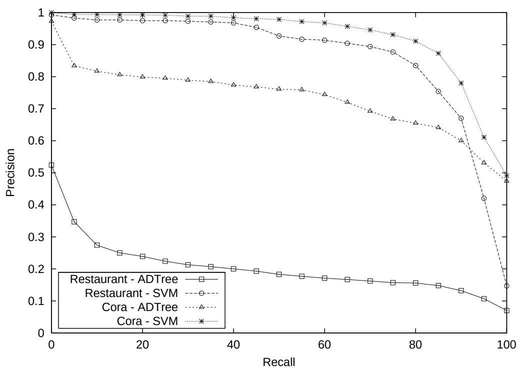

Figure 6: Duplicate detection results for *Restaurant* and *Cora* datasets using different record-level classifiers on limited training data

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 [Zadrozny and Elkan, 2001]. 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

Fellegi and Sunter [Fellegi and Sunter, 1969] 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 [Winkler, 1999]. That work was highly specialized for the domain of census records and used hand-tuned similarity measures.

McCallum et. al. introduced the efficient canopies clustering algorithm [McCallum *et al.*, 2000] for the task of matching scientific citations. Monge and Elkan developed the iterative merging algorithm based on the union-find data structure and showed the advantages of using a string distance metric that allows gaps [Monge and Elkan, 1997]. Cohen et. al. [Cohen *et al.*, 2000] 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 [Cohen and Richman, 2002]. Sarawagi and Bhamidipaty [Sarawagi and Bhamidipaty, 2002] and Tejada et. al. [Tejada *et al.*, 2002] developed systems that employ committee-based active learning methods for selecting informative record pairs to train the classifier that combines similarity estimates from multiple fields. In all of these approaches fixed-cost similarity metrics were used to compare individudual field values. 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 Conclusions and Future Work

Duplicate detection is an important problem for the data integration process, 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 particular domain. Our overall framework for duplicate detection integrates previous work on adaptive methods with learnable similarity measures, leading to improved results. Extending the metric learning approach to token-based similarity measures, such as vector-space cosine similarity, as well as developing more advanced models of character-based metrics are promising research directions that we are currently pursuing. Other avenues for future research include using learnable string metrics for other informartion integration tasks, such as ontology matching and wrapper generation.

7 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.

References

[Baeza-Yates and Ribeiro-Neto, 1999] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, 1999.

[Bilenko and Mooney, 2002] Mikhail Bilenko and Raymond J. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. TR AI 02-296, Artificial Intelligence Laboratory, U. of Texas at Austin, February 2002.

[Cohen and Richman, 2002] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. of KDD-2002*, Edmonton, Alberta, 2002.

[Cohen *et al.*, 2000] William W. Cohen, Henry Kautz, and David McAllester. Hardening soft information sources. In *Proc. of KDD-2000*, Boston, MA, August 2000.

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

[Freund and Mason, 1999] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proc. of ICML-99*, Bled, Slovenia, 1999.

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

[Hernández and Stolfo, 1995] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proc. of SIGMOD-95*, pages 127–138, San Jose, CA, May 1995.

[Joachims, 1999] Thorsten 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*, pp. 169–184. MIT Press, 1999.

[McCallum *et al.*, 2000] Andrew K. McCallum, Kamal Nigam, and Lyle Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of KDD-2000*, pp. 169–178, Boston, MA, August 2000.

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

[Nahm and Mooney, 2000] Un Yong Nahm and Raymond J. Mooney. Using information extraction to aid the discovery of prediction rules from texts. In *Proc. of KDD-2000 Workshop on Text Mining*, Boston, MA, August 2000.

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

[Ristad and Yianilos, 1998] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE PAMI*, 20(5), 1998.

[Sarawagi and Bhamidipaty, 2002] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proc. of KDD-2002*, Edmonton, Alberta, 2002.

[Tejada *et al.*, 2002] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. of KDD-2002*, Edmonton, Alberta, 2002.

[Vapnik, 1998] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[Winkler, 1999] William 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.

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

[Zadrozny and Elkan, 2001] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proc. of ICML-2001*, Williamstown, MA, 2001.