Copyright © 2003 IEEE. Reprinted from *IEEE Intelligent Systems*, vol. 18. no. 5. This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any mentioned products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank email message to pubs-permissions@ieee.org.

Adaptive Name Matching in Information Integration

Mikhail Bilenko and Raymond Mooney, *University of Texas at Austin*

William Cohen, Pradeep Ravikumar, and Stephen Fienberg, *Carnegie Mellon University*

Identifying approximately duplicate database records that refer to the same entity is essential for information integration. The authors compare and describe methods for combining and learning textual similarity measures for name matching.

When you combine information from heterogeneous information sources, you must identify data records that refer to equivalent entities. However, records that describe the same object might differ syntactically—for example, the same person can be referred to as "William Jefferson Clinton" and "bill clinton." Figure 1 presents more complex examples of duplicate records that are not identical.

Variations in representation across information sources can arise from differences in formats that store data, typographical and optical character recognition (OCR) errors, and abbreviations. Variations are particularly pronounced in data that's automatically extracted from Web pages and unstructured or semistructured documents, making the matching task essential for information integration on the Web.

Researchers have investigated the problem of identifying duplicate objects under several monikers, including record linkage, merge-purge, duplicate detection, database hardening, identity uncertainty, coreference resolution, and name matching. Such diversity reflects research in several areas: statistics, databases, digital libraries, natural language processing, and data mining. The sidebar summarizes various traditional approaches to name matching.

Our research explores approaches to the name-matching problem that improve accuracy. Particularly, we employ methods that adapt to a specific domain by combining multiple string similarity methods that capture different notions of similarity.

Static string similarity metrics

Most methods we discuss in the sidebar use metrics that measure the similarity between individual record fields. As Figure 1 suggests, individual record fields are often stored as strings. This means that functions that accurately measure two strings' similarity are important in duplicate identification. We examine a few effective and widely used metrics for measuring similarity.

Edit distance

An important class of such metrics are *edit distances*. Here, the distance between strings s and t is the cost of the best sequence of *edit operations* that converts s to t. For example, consider mapping the string s = "Willlaim" to t = "William" using these edit operations:

- *Copy* the next letter in s to the next position in t.
- *Insert* a new letter in t that does not appear in s.
- *Substitute* a different letter in t for the next letter in s.
- *Delete* the next letter in s; that is, don't copy it to t.

Table 1 shows one possible sequence of operations. (The vertical bar represents a "cursor" in s or t, indicating the next letter.) If the copy operation has cost zero and all other operations have cost one, this is the least expensive such sequence for s and t, so the edit distance between s and t is three.

A fairly efficient scheme exists for computing the lowest-cost edit sequence for these operations. The trick is to consider a slightly more complex function, $D(s, t, i, j)$, which is the edit distance between the first i letters in s and the first j letters in t. Let s_i denote the ith letter of s, and, similarly, let t_j be the jth letter of t. It's easy to see that we can define $D(s, t, i, j)$ recursively, where $D(s, t, 0, 0) = 0$ and

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

$$D(s,t,i,j) = \min \begin{cases} D(s,t,i-1,j-1) & \text{if } s_i = t_j \text{, and you copy } s_i \text{ to } t_j \\ D(s,t,i-1,j-1)+1 & \text{if you substitute } t_j \text{ for } s_i \\ D(s,t,i,j-1)+1 & \text{if you insert the letter } t_j \\ D(s,t,i-1,j)+1 & \text{if you delete the letter } s_i \end{cases} \quad (1)$$

We can evaluate this recursive definition efficiently using dynamic programming techniques. Specifically, for a fixed s and t, we can store the $D(s, t, i, j)$ values in a matrix that is filled in a particular order. The total computational effort for $D(s, t, |s|, |t|)$, the edit distance between s and t, is thus approximately $O(|s||t|)$.

Edit distance metrics are widely used—not only for text processing but also for biological sequence alignment[1]—and many variations are possible. The simple unit-cost metric just described is usually called *Levenshtein distance*. The *Needleman-Wunsch distance* is a natural extension to Levenshtein distance that introduces additional parameters defining each possible character substitution's cost and the cost of insertions and deletions. We can implement it straightforwardly by modifying Equation 1—replacing the second term of the min with something such as $D(s, t, i - 1, j - 1) + substitution\text{-}Cost(s_i, t_j)$. The *Smith-Waterman distance* lets us easily modify the metric to discount mismatching text at the beginning and the end of strings. The *affine gap* cost is another widely used variation that introduces two costs for insertion: one for inserting the first character and a second (usually lower) for inserting additional characters.

In information integration, Alvaro Monge and Charles Elkan performed several experiments in which they used an affine-cost variant of the Smith-Waterman distance to perform duplicate detection.[2,3]

The Jaro metric and variants

Another effective similarity metric is the *Jaro metric*, which is based on the number and order of common characters between two strings.[4–6] Given strings $s = a_1 \ldots a_K$ and $t = b_1 \ldots b_L$, define a character a_i in s to be "in common" with t if and only if there is a $b_j = a_i$ in t such that $i - H \le j \le i + H$, where $H = \min(|s|,|t|)/2$. Let $s' = a_1 \ldots a_k$ be the characters in s that are common with t (in the same order they appear in s), and let $t' = b_1 \ldots b_L$ be analogous. Then define a transposition for s', t' to be a position i such that $a_i \neq b_i$. Let $T_{s',t'}$ be one-half the number of transpositions for s' and t'. The Jaro metric for s and t is

$$Jaro(s,t) = \frac{1}{3} \cdot \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{s'} \right).$$

To better understand the intuition behind this metric, consider the matrix M in Figure 2, which compares the strings s = "WILLLAIM" and t = "WILLIAM." The boxed entries are the main diagonal, and $M(i, j) = 1$ if and only if the ith character of s equals the jth character of t. The Jaro metric is based on the number of characters in s that are in common with t. In terms of the matrix M of the figure, the ith character of s is in common with t if $M_{i,j} = 1$ for some entry (i, j) that is "sufficiently close" to M's main diagonal, where sufficiently close means that $|i - j| < \min(|s|, |t|)/2$ (shown in the matrix in bold).

William Winkler proposed a variant of the Jaro metric that also uses the length P of the longest common prefix of s and t.[6] Letting $P' = \max(P, 4)$, we define

$$Jaro\text{-}Winkler(s, t) = Jaro(s, t) + (P'/10) \cdot (1 - Jaro(s, t)).$$

This emphasizes matches in the first few characters. The Jaro and Jaro-Winkler metrics seem to be intended primarily for short strings (for example, personal first or last names).

Token-based and hybrid distances

In many situations, word order is unimportant. For instance, the strings "Ray Mooney" and "Mooney, Ray" are likely to be duplicates, even if they aren't close in edit distance. In such cases, we might convert the strings s and t to token multisets (where each token is a word) and consider similarity metrics on these multisets.

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)

(a)

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, pp. 395–410

(b)

Figure 1. Sample duplicate records from (a) a restaurant database and (b) a scientific citation database.

Table 1. An example of an edit-distance computation.

s	t	Operation
|Willlaim	|	
W|illlaim	W|	Copy "W"
Wi|llaim	Wi|	Copy "i"
Wil|llaim	Wil|	Copy "l"
Will|laim	Will|	Copy "l"
Willl|aim	Will|	Delete "l"
Willla|im	Willi|	Substitute "i" for "a"
Willlai|m	Willia|	Substitute "a" for "i"
Willlaim|	William|	Copy "m"

	W	I	L	L	I	A	M
W	**1**	**0**	**0**	0	0	0	0
I	**0**	**1**	**0**	**0**	1	0	0
L	**0**	**0**	**1**	**1**	**0**	0	0
L	0	**0**	**1**	**1**	**0**	**0**	0
L	0	0	**1**	**1**	**0**	**0**	**0**
A	0	0	0	**0**	**0**	**1**	**0**
I	0	1	0	0	**1**	**0**	**0**
M	0	0	0	0	0	**0**	**1**

Figure 2. The Jaro metric. The boxed entries are the main diagonal, and each bold character is in common with the string "WILLIAM" ("WILLLAIM").

One simple and often effective token-based metric is *Jaccard similarity*. Between the word sets S and T, Jaccard similarity is simply $(|S \cap T|)/(|S \cup T|)$. We can define *term frequency-inverse document frequency* (TF-IDF) or *cosine similarity*—which the information retrieval community widely uses—as

$$TF\text{-}IDF(S,T) = \sum_{w \in S \cap T} V(w,S) \cdot V(w,T),$$

Traditional Name-Matching Approaches

Record linkage—the task of matching equivalent records that differ syntactically—was first explored in the late 1950s and 1960s.[1] Ivan Fellegi and Alan Sunter's seminal paper—where they studied record linkage in the context of matching population records—provides a theoretical foundation for subsequent work on the problem.[2] They described several key insights that still lie at the base of many modern name-matching systems:

- You can represent every pair of records using a vector of features that describe similarity between individual record fields. Features can be Boolean (for example, last-name-matches), discrete (for example, first-n-characters-of-name-agree), or continuous (for example, string-edit-distance-between-first-names).
- The problem of identifying matching records can be viewed as the task of placing feature vectors for record pairs into three classes: links, nonlinks, and possible links. These correspond to equivalent, nonequivalent, and possibly equivalent (for example, requiring human review) record pairs, respectively.
- A system can perform record-pair classification by calculating the ratio $(P(\gamma|M))/(P(\gamma|U))$ for each candidate record pair, where γ is a feature vector for the pair and $P(\gamma|M)$ and $P(\gamma|U)$ are the probabilities of observing that feature vector for a matched and nonmatched pair, respectively. Two thresholds based on desired error levels—T_μ and T_λ—optimally separate the ratio values for equivalent, possibly equivalent, and nonequivalent record pairs.
- When no training data in the form of duplicate and nonduplicate record pairs is available, name-matching can be unsupervised, where conditional probabilities for feature values are estimated using field value frequencies.
- Because most record pairs are clearly nonduplicates, you needn't consider them for matching; *blocking* databases so that only records in blocks are compared significantly improves efficiency.

The first four insights lay the groundwork for accurate record pair matching, while the fifth provides for efficiently processing large databases. We can describe subsequent name-matching research in terms of improvements in those two directions.

Several methods address the computational cost of name matching and follow the spirit of the blocking mechanism the Fellegi-Sunter theory suggests. The *sorted neighborhood* method sorts the database using multiple keys to obtain record blocks ("windows") in which candidates for matching lie.[3] Alternatively, the *canopies* method uses a computationally cheap and general string similarity metric such as *term frequency-inverse data frequency* (TF-IDF) cosine similarity to create overlapping record clusters that contain possible matching pairs.[4]

We can roughly categorize methods for improving matching accuracy by how much human expertise they require and the extent to which they use machine learning and probabilistic methods. On one end of this spectrum are rule-based methods based on equational theory that require a human expert to specify the conditions for records to be equivalent in a declarative rule language.[3,5–7] Such conditions might involve multiple string similarity metrics (for example, the string edit distance being less than a threshold value), domain-specific comparisons (equality of nicknames and full first names), and inferred knowledge (geographic proximity based on zip codes). Although the rule-based approach can lead to high accuracy after meticulous, domain-specific tuning, its human cost tends to be high and therefore impractical for large databases.

Unlike the rule-based approach, probabilistic methods developed following the Fellegi-Sunter framework obviate the need to involve human domain expertise by using unsupervised machine learning methods. We can employ the powerful *expectation maximization* algorithm to classify record pairs into the three classes we specified without any training data on the basis of the database's statistical properties.[8] In an iterative procedure, EM estimates the probability that the records match for each pair of records. We can add additional constraints to the standard EM algorithm to enforce one-to-one matching when records are being matched across two databases, thereby avoiding spurious multiple matches.[9]

An alternative unsupervised approach to domain-independent matching assumes that data is stored in databases as natural language text and treats the matching task as an information retrieval problem.[10] This approach achieves domain independence through *normalization*, which uses preprocessing such as case conversion and stemming, then employs cosine similarity in the vector space created using the TF-IDF weighting scheme (see the main text). This approach often works well for databases where records can be meaningfully represented as natural text strings. An alternative approach to dealing with such databases is to separate string records into individual fields that represent atomic information units—for

where $TF_{w,S}$ is the frequency of word w in S, IDF_w is the inverse of the fraction of names in the corpus that contain w, $V(w, S) = \log(TF_{w,S} + 1) \cdot \log(IDF_w)$, and

$$V(w,S) = V(w,S) / \sqrt{\sum_w V(w,S)^2} .$$

For name matching, you might collect the statistics used to compute IDF_w from the complete corpus of names to be matched.

TF-IDF distance is attractive in that it weights agreement on rare terms more heavily than agreement on more common terms. This means that "Ray Mooney" and "Wray Mooney" will be considered more similar than, say, "Ray Mooney" and "Ray Charles." Perhaps this is why TF-IDF often (but not always) performs better than methods that are insensitive to individual tokens' frequency, such as the Jaccard similarity.

Some interesting commonalities exist between the TF-IDF weighting scheme and an unsupervised matching approach Fellegi and Sunter proposed—ranking pairs s, t by the odds ratio $\log((\Pr(|s, t|M))/(\Pr(|s, t|U)))$, where M is the class of matched pairs and U is the class of unmatched pairs. Under a plausible series of assumptions, we can approximate the incremental score for the odds ratio associated with the event "s and t both agree in containing word w" using $\log(IDF_w)$.

We can also combine token-based and string-based methods. In previous work, we described a "soft" version of TF-IDF in which similarity is affected not only by tokens that appear in both S and T but also by tokens in S

example, to parse a citation record into separate fields such as *author*, *title*, *venue*, and so on. Hidden Markov models are particularly successful for this task if they receive sufficient training data in the form of segmented strings.[11,12]

Another avenue for using supervised learning to improve name matching relies on creating a relational probabilistic model for the domain. This involves constructing a generative model for individual fields and using a Markov chain Monte Carlo procedure to obtain the matching decisions.[13] This approach allows for capturing the different matching decisions' interdependence. This is useful for databases that contain several matching records, such as bibliographies of citations to scientific papers. Accounting for the distributed nature of matching decision making in databases with many equivalent records is also central to the *database hardening* approach, which formalizes name matching as a mathematical optimization problem and suggests a greedy algorithm for obtaining the best global record matching using a graph of similarity values between records.[14]

Recently, researchers have proposed machine-learning methods that use supervision in the form of matched and unmatched record pairs to train classifiers to distinguish between them. This includes those methods that try to select the most informative record pairs for human labeling to produce maximum accuracy improvements.[15] The main text describes our recent work using training data in the form of matched and unmatched record pairs to train an algorithm for classifying record pairs as duplicate and nonduplicate.

References

1. H.B. Newcombe et al., "Automatic Linkage of Vital Records," *Science*, vol. 130, no. 3381, Oct. 1959, pp. 954–959.
2. I.P. Fellegi and A.B. Sunter, "A Theory for Record Linkage," *J. American Statistical Assoc.*, vol. 64, no. 328, Dec. 1969, pp. 1183–1210.
3. M.A. Hernández and S.J. Stolfo, "The Merge/Purge Problem for Large Databases," *Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data* (SIGMOD 95), ACM Press, 1995, pp. 127–138.
4. A.K. McCallum, K. Nigam, and L. Ungar, "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching," *Proc. 6th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2000), ACM Press, 2000, pp. 169–178.
5. H. Galhardas et al., "AJAX: An Extensible Data-Cleaning Tool," *Proc. 2000 ACM SIGMOD Int'l Conf. Management of Data* (SIGMOD 00), ACM Press, 2000, p. 590.
6. H. Galhardas et al., "Declarative Data Cleaning: Language, Model, and Algorithms," *Proc. 27th Int'l Conf. Very Large Databases* (VLDB 2001), Morgan Kaufmann, 2001, pp. 371–380.
7. M.-L. Lee, T.W. Ling, and W.L. Low, "Intelliclean: A Knowledge-Based Intelligent Data Cleaner," *Proc. 6th Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2000), ACM Press, 2000, pp. 290–294.
8. W.E. Winkler, "Using the EM Algorithm for Weight Computation in the Fellegi-Sunter Model of Record Linkage," *Proc. Section on Survey Research Methods*, American Statistical Assoc., 1988, pp. 667–671.
9. W.E. Winkler, *Advanced Methods for Record Linkage*, tech. report, Statistical Research Division, US Census Bureau, 1994.
10. W.W. Cohen, "Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity," *Proc. ACM SIGMOD Int'l Conf. Management of Data* (SIGMOD 98), ACM Press, 1998, pp. 201–212.
11. K. Seymore, A.K. McCallum, and R. Rosenfeld, "Learning Hidden Markov Model Structure for Information Extraction," *Papers from the 16th Nat'l Conf. Artificial Intelligence* (AAAI 99) *Workshop Machine Learning for Information Extraction*, AAAI Press, 1999, pp. 37–42.
12. T. Churches et al., "Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models," *Medical Informatics and Decision Making*, vol. 2, no. 9, 13 Dec. 2002; www.biomedcentral.com/1472-6947/2/9/abstract.
13. H. Pasula et al., "Identity Uncertainty and Citation Matching," *Advances in Neural Information Processing Systems 15*, MIT Press, 2003.
14. W.W. Cohen, H. Kautz, and D. McAllester, "Hardening Soft Information Sources," *Proc. 6th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2000), ACM Press, 2000, pp. 255–259.
15. S. Tejada, C.A. Knoblock, and S. Minton, "Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification," *Proc. 8th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2002), ACM Press, 2002, pp. 350–359.

such that a similar token appears in T. Let *sim* be a secondary similarity function that performs well on short strings (such as Jaro-Winkler). Let $CLOSE(\theta, S, T)$ be the set of words $w \in S$ such that some $v \in T$ exists such that $sim(w, v) > \theta$, and for $w \in CLOSE(\theta, S, T)$, let $N(w, T) = \max_{v \in T} sim(w, v)$. We define

$$SoftTF\text{-}IDF(S,T) = \sum_{w \in CLOSE(\theta,S,T)} V(w,S) \cdot V(w,T) \cdot N(w,T).$$

Relative performance of the static similarity metrics

To indicate these distance functions' relative effectiveness, we evaluated them using the set of record-matching problems in Table 2. Most of these problems have been described elsewhere in the literature.[2,7,8] In all these data sets, if a record had multiple fields, we concentrated them to form a single string. The Census data set is a synthetic, census-like data set from which we used only textual fields (last name, first name, middle initial, house number, and street).

To evaluate distance functions on a data set, we ranked by distance all candidate pairs an appropriate blocking algorithm generated (see the sidebar for a definition of blocking algorithms). (For token-based distance metrics, the blocking algorithm generates all pairs that contain at least one common token. For other distance metrics, it generates all pairs containing at least one common character 4-gram.) In practice, we would examine this ranking and choose a suitable threshold θ. We could consider all pairs more similar than this threshold to be

Table 2. The data sets used in experiments.

Name	No. of strings	No. of tokens
Animal[7]	5,709	30,006
Bird1[7]	377	1,977
Bird2[7]	982	4,905
Bird3[7]	38	188
Bird4[7]	719	4,618
Business[7]	2,139	10,526
Game[7]	911	5,060
Park[7]	654	3,425
Restaurant[8]	863	10,846
UcdFolks[2]	90	454
Census*	841	5,765

*Data obtained from personal communication with William Winkler

Table 3. The performance of various matching methods on the Census data set.

Method	Maximum F1 score	Average precision
TF-IDF	0.518	0.369
Jaccard	0.567	0.402
SoftTF-IDF	0.685	0.782
Jaro	0.728	0.789
Levenshtein	0.865	0.925

matches and all pairs less similar to be nonmatches.

Without committing to any particular threshold, several plausible ways exist to evaluate such a ranking; most are based on recall and precision measures. If we fix a threshold θ, recall will be the fraction of the set of all correct matches that have a similarity greater than θ. Conversely, precision is the fraction of correct matches having a similarity greater than θ. As the similarity threshold θ is reduced, precision will typically decrease and recall will increase. So, we can summarize the ranking as a single monotonically decreasing curve in two dimensions.

Figure 3 presents recall-precision curves that are averages across all the benchmark problems. Specifically, we use *interpolated precision* at 11 evenly spaced recall points. Interpolated precision at recall r is the maximum precision obtained at any point with recall greater than or equal to r. Figure 3a shows results for edit distance and Jaro variants. Figure 3b shows results for some token-based and hybrid measures. Of the edit-distance-like methods, Monge-Elkan performs best on average, and SoftTF-IDF performs best overall.

To summarize a ranking's value as a single number, we can use the maximum F1 score. The F1 score for a threshold is the harmonic mean of recall and precision (that is, $F1 = 2pr/(p + r)$, where p is precision and r is recall). The maximum F1 score is simply the maximum value of F1 that is obtained for any threshold. Figure 3c details SoftTF-IDF's performance compared to numerous other measures that perform well on average. In this plot, each point is a data set, positioned so that its maximum F1 score for SoftTF-IDF is the y-axis position and the maximum F1 score for some other method is the x-axis position. So, points above the line $y = x$ indicate better performance for SoftTF-IDF.

Segmenting the record also means that we can represent records' similarity with a feature vector instead of a single similarity measurement.

Learning to combine field similarities

Figure 3 focuses on results that are averaged across several benchmark data sets. However, for any individual problem, performance can differ markedly from average performance, as Figure 3c indicates. Table 3 also emphasizes this fact.

Table 3 shows that the Census data set behaves quite differently from most of the others in Table 2. (In addition to the F1 measure, the table shows average precision, which is simply precision averaged over every threshold θ that is just below a correct pair.) Perhaps owing to the high incidence of misspellings, none of the token-based metrics perform well on Census. Unusually, Jaccard performs about as well as any of the more complex token-based metrics. This could be because Census includes several households with a moderate number of individuals—for instance, a family of seven Mosqueras resides at one address and a family of five Hoerrlings at another—which reduces the value of weighting rare terms heavily. The hybrid SoftTF-IDF performs somewhat better but is still worse than the pure string-based methods. The Jaro metric performs surprisingly well given that it seems designed for short strings, while the Census data contains first name, last name, middle initial, house number, and street name appended together. The best off-the-shelf distance metric on Census, by far, is the Levenshtein metric—the simplest of the edit-distance approaches and the one that performs worst on average.

Table 2 illustrates a dilemma: even methods that have been tuned and tested on many previous matching problems can perform poorly on new and different matching problems. This is a fundamental limitation of static similarity functions, which by nature can include no special knowledge of the specific problem at hand.

To solve this problem, we must introduce some knowledge of the problem into the similarity function we're using. We can do this by treating each record as a set of fields rather than a single, long-string field and then aggregating distances across these fields. For instance, consider breaking the Census data into the original fields (first name, last name, middle initial, house number, and street name) and measuring the similarity between records as the average similarity between fields. This small change has numerous implications. For instance, we must no longer implicitly align the different fields in measuring similarity, and short fields (such as house number) receive the same weight as longer fields. Table 4 compares this method's performance (labeled AVG) with computing similarity based on the same record encoded as a single string (labeled "Single string"). For five of the six comparisons, performance improved.

Segmenting the record also means that we can represent records' similarity with a feature vector instead of a single similarity measurement. If some pairs are available that have been labeled as matching and nonmatching, we can use learning methods to adaptively find a combined distance function that's most appropriate for a particular problem.

In another experiment, we represented record pairs as feature vectors, using as features the distances between corresponding fields. We trained a binary support vector machine classifier (using the SVM*light* software package[9]) using these feature vectors, then used the learned classifier's confidence in the *match* class as a new distance metric. Table 3 shows this method's performance evaluated on unseen pairs using threefold cross validation (in the rows labeled SVM). For five of the six comparisons, performance improved over the simple averaging of the distances, and performance improved in all cases over the single-string distance computation.

Learnable string similarity

The learning method we just described is limited. Although it can determine how to weight a fixed set of static string similarity measures, it can't modify the similarity measures each field uses. However, because an estimate of similarity between strings can vary significantly depending on the domain for each field under consideration, static similarity measures could fail to accurately estimate string distance. When syntactic variations occur owing to systematic typographical or OCR errors, certain characters can be consistently replaced with others or omitted. Certain abbreviations are also common to some domains; for example, Street is frequently abbreviated to St. in addresses. So, precise similarity computations require adapting string similarity metrics for each database field with respect to the particular data domain.

We propose an adaptive version of edit distance with affine gaps such as those we previously described. Because different edit operations vary in significance in different domains, adapting string edit distance to a particular text data type necessitates assigning individual weights to edit operations. Each operation's weight must capture the likelihood with which we can apply it to align equivalent strings in a particular domain's context. For example, the "substituting '-' with '/'" edit operation should have a low weight for a text field that describes a phone number because it captures a common difference between equivalent phone numbers. Conversely, the "deleting 'q'" operation should be significant for comparing first names because it concerns a rare letter that's usually at the beginning of a name (for example, Quincy or Quentin). So, we probably wouldn't apply it to form abbreviations and diminutives.

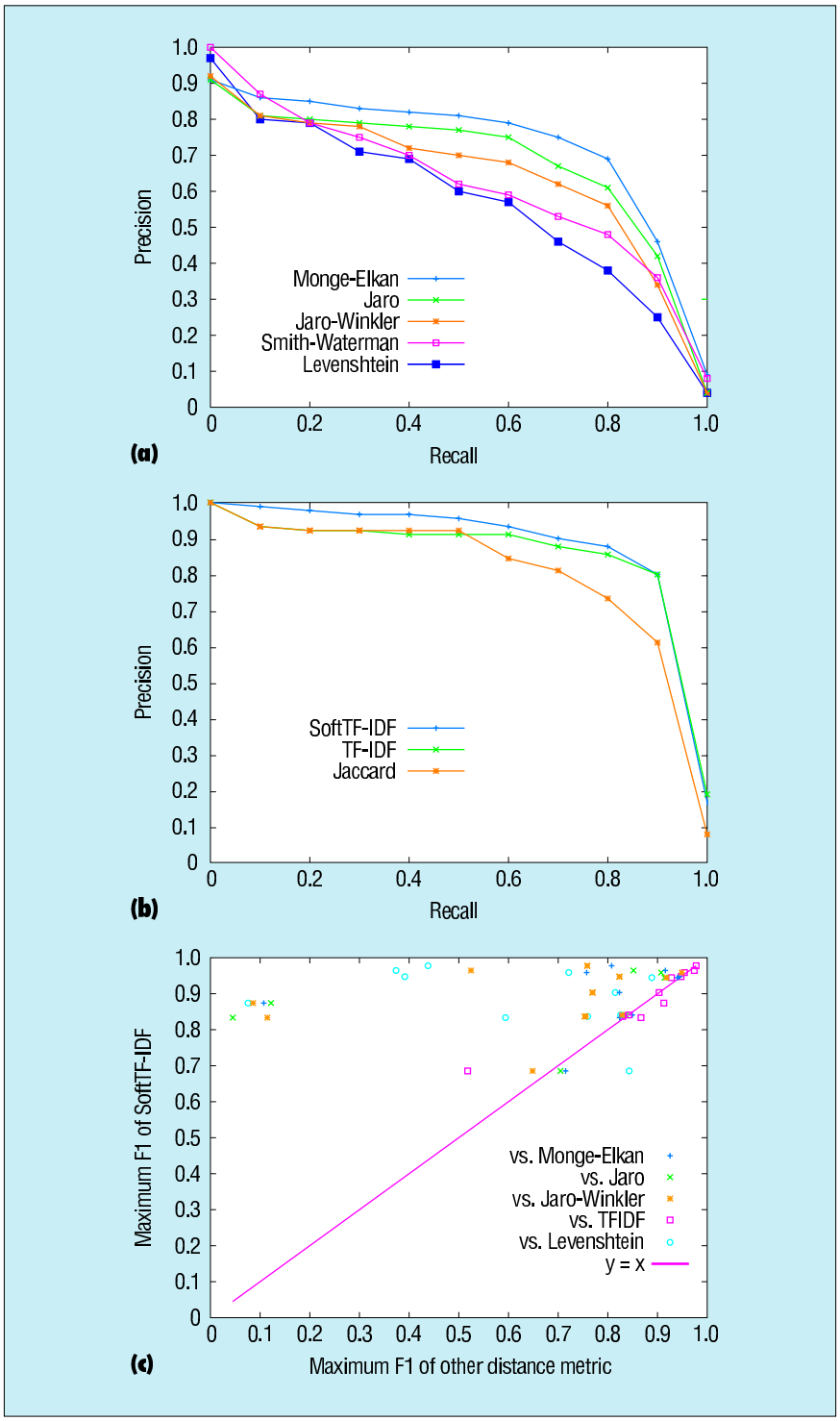


Figure 3. Average recall precision curves showing the relative performance of (a) edit-distance measures; (b) token-based measures; and (c) several high-average-score measures compared to SoftTF-IDF.

Likewise, we should be able to adapt the costs of starting a gap, extending a gap, and continuous substitution to a particular domain. Long sequences of contiguous insertions and deletions are common for certain data types; for example, conference titles in scientific citations are often abbreviated. In other domains, such as telephone numbers,

Table 4. Performance of various structure-exploiting methods on the Census data set.

String metric	Field aggregation method	Maximum F1	Average precision
SoftTF-IDF	SVM	0.792	0.830
	AVG	0.803	0.810
	Single string	0.685	0.782
Jaro	SVM	0.917	0.932
	AVG	0.897	0.922
	Single string	0.728	0.789
Levenshtein	SVM	0.890	0.928
	AVG	0.870	0.920
	Single string	0.865	0.925

gaps are infrequent and short and therefore must incur high cost.

Our model for the edit distance's adaptive version with affine gaps is similar to prior work by Eric Ristad and Peter Yianilos, who developed a learnable model for Levenshtein distance.[10] We can represent edit distance with affine gaps using a hidden Markov model[11] that produces character alignments between two strings, possibly including gaps. This model generates a particular alignment of two strings as a sequence of character pairs. Each pair is either an aligned pair of characters from the two strings or a character aligned to a gap corresponding to an insertion or a deletion.

The distance between two strings in this model corresponds to the probability of generating the most likely alignment between the strings' characters. We can compute this using dynamic programming in standard forward and backward algorithms in time proportional to the strings' lengths.[11] Given a corpus of matched strings corresponding to duplicate pairs, we can train the model using a variant of the Baum-Welch algorithm,[12] which is an expectation maximization (EM) procedure for learning generative models' parameters. We can use the trained model to estimate distance between strings by computing the probability of generating the aligned pair of strings. Strings that are aligned with a high probability are similar, while strings for which the optimal alignment has a low probability are dissimilar.

> The parameters our training algorithm obtained capture certain domain properties that allow more accurate similarity computations.

To evaluate the usefulness of adapting string similarity measures to a specific domain, we compared learnable and fixed-cost distances with affine gaps to identify equivalent field values. We conducted our experiments on four single-field scientific citation data sets from the CiteSeer digital library. The Reasoning, Face, Reinforcement, and Constraint data sets contain citations for corresponding computer science areas: automated reasoning, face recognition, reinforcement learning, and constraint satisfaction. Also, we tested our approach on the two most meaningful fields from the Restaurant data set—name and address. The matching fields for these two data sets are noisy because we assume that matching records have duplicate individual fields and vice versa. This assumption can be incorrect, leading to unavoidable testing errors. For example, two distinct restaurants can exist in different cities with a variation of the same name, while in these trials we would consider the two name fields nonmatching because they correspond to nonduplicate records.

We randomly split every data set into two folds to cross validate each experimental run by grouping equivalent records and assigning them to a random fold. We report all results over 20 random splits, where, for each split, we used the two folds alternately for training and testing. In the training phase, we used matched record pairs from the training fold to learn the parameters of the learnable edit distance just described. In the testing phase, we computed the similarity between records in the testing fold and merged most similar records iteratively. After every merged step, we computed the transitive closure of matched records to create matched record clusters.

Table 5 summarizes our results for field-level duplicate detection experiments. Each entry contains the average of maximum F1 scores over the 40 evaluated folds. These results demonstrate that despite the noise, learned affine edit distance can outperform nontrained edit distance when detecting duplicate database fields. Visually inspecting the learned parameter values shows us that the parameters our training algorithm obtained 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." This captures the fact that a common cause of street name duplicates is abbreviations from Street to St. Overall, the results indicate that learnable string edit distance outperforms the static variant and leads to higher name-matching accuracy when training pairs in the form of duplicate records are available.

Identifying distinct records that refer to the same entity is an important information-integration subproblem that has been stud-

Table 5. Maximum F1 scores 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

ied for over 40 years. Most recent work in the area has focused on using the latest machine-learning techniques, such as EM and SVMs, to learn accurate matching functions. We have described how such techniques can be used to accurately identify equivalent records when integrating data from multiple, heterogenous sources. ❐

Acknowledgments

The National Science Foundation partially supported this article's preparation under grants IIS-0117308 and EIA-0131884. It was also supported by a contract from the Army Research Office to the Center for Computer and Communications Security with Carnegie Mellon University and by a faculty fellowship from IBM.

References

1. R. Durban et al., *Biological Sequence Analysis—Probabilistic Models of Proteins and Nucleic Acids*, Cambridge Univ. Press, 1998.

2. A. Monge and C. Elkan, "The Field-Matching Problem: Algorithm and Applications," *Proc. 2nd ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, 1996, pp. 267–270.

3. A. Monge and C. Elkan, "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records," *Proc. SIGMOD Workshop Data Mining and Knowledge Discovery*, ACM Press, 1997, pp. 267–270.

4. M.A. Jaro, "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida," *J. Am. Statistical Assoc.*, vol. 84, no. 406, June 1989, pp. 414–420.

5. M.A. Jaro, "Probabilistic Linkage of Large Public Health Data Files," *Statistics in Medicine*, vol. 14, nos. 5–7, Mar./Apr. 1995, pp. 491–498.

6. W.E. Winkler, "The State of Record Linkage and Current Research Problems," *Statistics of Income Division*, Internal Revenue Service Publication R99/04, 1999; www.census.gov/srd/www/byname.html.

7. W.W. Cohen, "Data Integration Using Similarity Joins and a Word-Based Information Representation Language," *ACM Trans. Information Systems*, vol. 18, no. 3, July 2000, pp. 288–321.

8. S. Tejada, C.A. Knoblock, and S. Minton, "Learning Object Identification Rules for Information Integration," *Information Systems*, vol. 26, no. 8, Dec. 2001, pp. 607–633.

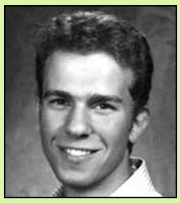
9. T. Joachims, *Learning to Classify Text Using Support Vector Machines*, Kluwer, 2002.

10. E.S. Ristad and P.N. Yianilos, "Learning String-Edit Distance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, May 1998, pp. 522–532.

11. L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, vol. 77, no. 2, Feb. 1989, pp. 257–286.

12. M. Bilenko and R.J. Mooney, "Adaptive Duplicate Detection Using Learnable String Similarity Measures," *Proc. 9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining* (KDD 2003), ACM Press, 2003, pp. 39–48.

The Authors

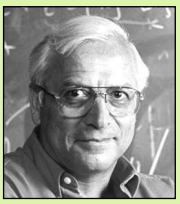


Mikhail Bilenko is a PhD student in the Department of Computer Sciences at the University of Texas at Austin. His research interests include semi-supervised learning, text mining, clustering, and information integration. He received his MSCS from the University of Texas at Austin. Contact him at the Dept. of Computer Sciences, Univ. of Texas at Austin, 1 University Station C0500, Austin, TX 78712-1188; mbilenko@cs.utexas.edu; www.cs.utexas.edu/users/mbilenko.

Raymond Mooney is a professor of computer sciences at the University of Texas at Austin. His research interests include text mining, learning for natural language processing, information extraction, text categorization, recommender systems, inductive-logic programming, and semisupervised clustering. He received his PhD in computer science from the University of Illinois at Urbana/Champaign. He is a member of the ACM, AAAI, Association for Computational Linguistics, and Cognitive Science Society. Contact him at the Dept. of Computer Sciences, Univ. of Texas at Austin, 1 University Station C0500, Austin, TX 78712-0233; mooney@cs.utexas.edu; www.cs.utexas.edu/users/mooney.

William Cohen is an associate research professor at Carnegie Mellon University's Center for Automated Learning and Discovery and an action editor for the *Journal of Machine Learning Research*. His research interests include information integration and machine learning, particularly text categorization, information extraction, and learning from large data sets. He received his PhD in computer science from Rutgers University. Contact him at the Center for Automated Learning & Discovery, Carnegie Mellon Univ., 5000 Forbes Ave., Pittsburgh, PA 15213; wcohen@cs.cmu.edu; www.cs.cmu.edu/~wcohen.

Pradeep Ravikumar is a PhD student at Carnegie Mellon University's Center for Automated Learning and Discovery at the School of Computer Science. His research interests include information integration, disclosure limitation, and machine learning. He received his Bachelor of Technology in computer science and engineering from the Indian Institute of Technology, Bombay. Contact him at Wean Hall, Rm. 4212, School of Computer Science, Carnegie Mellon Univ., 5000 Forbes Ave., Pittsburgh, PA 15213; pradeepr@cs.cmu.edu; www.cs.cmu.edu/~pradeepr.



Stephen Fienberg is a Maurice Falk University professor of statistics and social science in the Department of Statistics, the Center for Automated Learning and Discovery, and the Center for Computer and Communications Security at Carnegie Mellon University. His research interests include analysis of categorical data, Bayesian approaches to confidentiality and data disclosure, causation, foundations of statistical inference, the history of statistics, sample surveys and randomized experiments, statistics and the law, and inference for multiple-media data. He received his PhD in statistics from Harvard University. He is a fellow of the American Association for the Advancement of Science, the American Statistical Association, and the Institute of Mathematical Statistics, and is an elected member of the National Academy of Sciences. Contact him at 132G Baker Hall, Dept. of Statistics, Carnegie Mellon Univ., Pittsburgh, PA 15213; fienberg@stat.cmu.edu; www.stat.cmu.edu/~fienberg.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.