A Comparison of String Metrics for Matching Names and Records

William W. Cohen
Center for Automated Learning and Discovery,
School of Computer Science,
Carnegie Mellon University
wcohen@wcohen.com

Pradeep Ravikumar
Center for Automated Learning and Discovery,
School of Computer Science,
Carnegie Mellon University
pradeepr@cs.cmu.edu

Stephen E. Fienberg
Department of Statistics,
Center for Computer & Communications Security,
& Center for Automated Learning & Discovery
Carnegie Mellon University
fienberg@stat.cmu.edu

Abstract

We describe an open-source Java toolkit of methods for matching names and records. We summarize results obtained from using various string distance metrics on the task of matching entity names. These metrics include distance functions proposed by several different communities, such as edit-distance metrics, fast heuristic string comparators, token-based distance metrics, and hybrid methods. We then describe an extension to the toolkit which allows records to be compared. We discuss some issues involved in performing a similar comparision for record-matching techniques, and finally present results for some baseline record-matching algorithms that aggregate string comparisons between fields.

Introduction

The task of matching entity names has been explored by a number of communities, including statistics, databases, and artificial intelligence. Each community has formulated the problem differently, and different techniques have been proposed.

In statistics, a long line of research has been conducted in *probabilistic record linkage*, largely based on the seminal paper by Fellegi and Sunter (1969). This paper formulates entity matching as a classification problem, where the basic goal is to classify entity pairs as matching or nonmatching. Fellegi and Sunter propose using largely unsupervised methods for this task, based on a feature-based representation of pairs which is manually designed and to some extent problem-specific. These proposals have been, by and large, adopted by subsequent researchers, often with elaborations of the underlying statistical model (Jaro 1989; 1995; Winkler 1999; Larsen 1999; Belin & Rubin 1997). These methods have been used to match individuals and/or families between samples and censuses, e.g., in evaluation of the coverage of the U.S. decennial census; or between administrative records and survey data bases, e.g., in the creation of an anonymized research data base combining tax information from the Internal Revenue Service and data from the Current Population Survey.

In the database community, some work on record matching has been based on knowledge-intensive approaches (Hernandez & Stolfo 1995; Galhardas *et al.* 2000; Raman & Hellerstein 2001). The use of string-edit distances as a general-purpose record matching scheme was proposed by Monge and Elkan (1997; 1996), and in previous work, we proposed use of the TFIDF distance metric for the same purpose (Cohen 2000). In the AI community, supervised learning has been used for learning the parameters of string-edit distance metrics (Ristad & Yianilos 1998; Bilenko & Mooney 2002) and combining the results of different distance functions (Tejada, Knoblock, & Minton 2001; Cohen & Richman 2002; Bilenko & Mooney 2002). More recently, probabilistic object identification methods have been adapted to matching tasks (Pasula *et al.* 2002). In these communities there has been more emphasis on developing autonomous matching techniques which require little or or no configuration for a new task, and less emphasis on developing "toolkits" of methods that can be applied to new tasks by experts.

Recently, we have begun implementing an open-source, Java toolkit of name-matching methods (Cohen & Ravikumar 2003) that includes a variety of different techniques. In previous work (Cohen, Ravikumar, & Fienberg 2003), we used this toolkit to conduct a comparison of several string distances on the tasks of matching and clustering lists of entity names. In addition to evaluating existing string-distance methods, we also proposed some new ones, including a hybrid of cosine similarity and the Jaro-Winkler method (Winkler 1999), which performed well on many of our benchmark problems.

These previous experiments, while similar to previous experiments in the database and AI communities, represent a departure from the usual assumptions made in statistics. In statistics, databases tend to have more structure and specification, by design. Thus the statistical literature on probabilistic record linkage represents pairs of entities not by pairs of strings, but by vectors of "match features" such as names and categories for variables in survey databases. In this paper we will review the previous experiments with matching individual strings, and then discuss some recent, preliminary experiments in extending our toolkit to matching structured objects (i.e., records).

The SecondString Toolkit

Overall Architecture

SecondString is an open-source Java toolkit of name-matching methods. One fundamental type of object in SecondString is a *distance function*. A *distance function* maps a pair of strings s and t to a real number r, where a smaller value of r indicates greater similarity between s and t. Since SecondString is designed to support learnable distance functions, a distance function is always produced by a *distance function learner*. A *distance function learner* can be "trained" in any of two ways:

- It can observe a set of strings from the distribution of strings to be matched. (We call this step *string observation*).
- It can be presented with a *pool* of unlabeled pairs of strings, some of which must be matched. The learner may then query an associated *distance function teacher* for labels for pairs in the pool. In controlled experiments the teacher answers queries using pre-labeled pairs; in real-world settings, it would ask a user for labels. The teacher may also refuse to answer a query. (We call this step *active learning*).

At any point in the learning process, the learner can be asked to produce a *distance function*—which will presumably be trained on all evidence available so far. Thus the architecture supports a range of types of learning, including unsupervised, semi-supervised, supervised, batch and incremental.

Although designed to support learning, SecondString also supports non-adaptive matching methods by including a degenerate distance function "learner" that simply produces a particular constant distance function. This allows non-adaptive methods (e.g., Levenstein edit distance) to be easily evaluated side-by-side with learned methods.

Implemented Distance Functions

SecondString supports a large number of non-adaptive distance functions, some of which are listed below. For more details, the reader is referred to our previous paper (Cohen, Ravikumar, & Fienberg 2003).

SecondString supports a range of metrics based on *edit distance*, including *Levenstein distance,* which assigns a unit cost to all edit operations); and the *Monge-Elkan* distance function (Monge & Elkan 1996), a well-tuned affine variant of the Smith-Waterman distance function (Durban *et al.* 1998). It also supports the *Jaro* metric (Jaro 1995; 1989), a metric widely used in the record-linkage community, with and without a variation due to Winkler (1999). Briefly, for two strings s and t, let s' be the characters in s that are "common with" t, and let t' be analogous; roughly speaking, a character a in s is "in common" with t if the same character a appears in about the place in t. Let $T_{s,t}$ measure the number of transpositions of characters in s' relative to t'. The Jaro similarity metric for s and t is

$$Jaro(s,t) = \frac{1}{3} \cdot \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{2|s'|} \right)$$

and the Winkler variant modifies this by slightly improving the weight of poorly matching pairs s, t that share a long common prefix (Cohen, Ravikumar, & Fienberg 2003).

SecondString also supports a number of token-based distance metrics, which are defined by considering two strings s and t to be multisets of words (or tokens). The *Jaccard similarity* between the word sets S and T is simply $\frac{|S\cap T|}{|S\cup T|}$. *TFIDF* or *cosine similarity* is another measure, widely used in the information retrieval community. Like Jaccard, the TFIDF scheme depends on common terms, but terms are weighted; these weights are larger for words w that are rare in the collection of strings from which s and t were drawn. (Acquiring these weights is an example of "string observation".) We have also implemented token-based distance metrics based on Jensen-Shannon distance (Dagan, Lee, & Pereira 1999) with various smoothing methods, and a simplified form of Fellegi and Sunter's method (Fellegi & Sunter 1969), called SFS below.

SecondString also supports some hybrid distance functions, which combine token-based and string-based matching schemes. In addition to a variant of Monge and Elkan's "recursive matching scheme", we have implemented a "soft" version of TFIDF, in which similar tokens are considered as well as tokens in $S \cap T$. Let sim' be a secondary similarity function. Let $CLOSE(\theta, S, T)$ be the set of words $u \in S$ such that there is some $v \in T$ such that $sim'(u,v) > \theta$, and for $u \in CLOSE(\theta,S,T)$, let $N(u,T) = \max_{v \in T} sim'(u,v)$. We define

$$SoftTFIDF(S,T) = \sum_{u \in CLOSE(\theta,S,T)} wt(u,S) \cdot wt(v,T) \cdot N(w,T)$$

where $wt(u,S)$ is the TFIDF weight of word u in S. In the experiments, we used Jaro-Winkler as a secondary distance sim' and used $\theta = 0.9$.

SecondString also supports tools for systematic experimentation, and tools for "blocking", or finding plausible pairs of names to match (Cohen, Ravikumar, & Fienberg 2003).

Experiments with SecondString

The data used to evaluate these methods is shown in Table 1. Most been described elsewhere in the literature. The Census dataset is a synthetic, census-like dataset, from which only textual fields were used (last name, first name, middle initial, house number, and street).

To evaluate a method on a dataset, we ranked by distance all candidate pairs from the appropriate blocking algorithm. We computed the non-interpolated average precision of this ranking, the maximum F1 score of the ranking, and also interpolated precision at the eleven recall levels 0.0, 0.1, …, 0.9, 1.0. The *non-interpolated average precision* of a ranking containing N pairs for a task with m correct matches is $\frac{1}{m}\sum_{r=1}^{N} \frac{c(i)\delta(i)}{i}$, where $c(i)$ is the number of correct pairs ranked before position i, and $\delta(i) = 1$ if the pair at rank i is correct and 0 otherwise. *Interpolated precision* at recall r is the $\max_i \frac{c(i)}{i}$, where the max is taken over all ranks i such that $\frac{c(i)}{m} \geq r$.

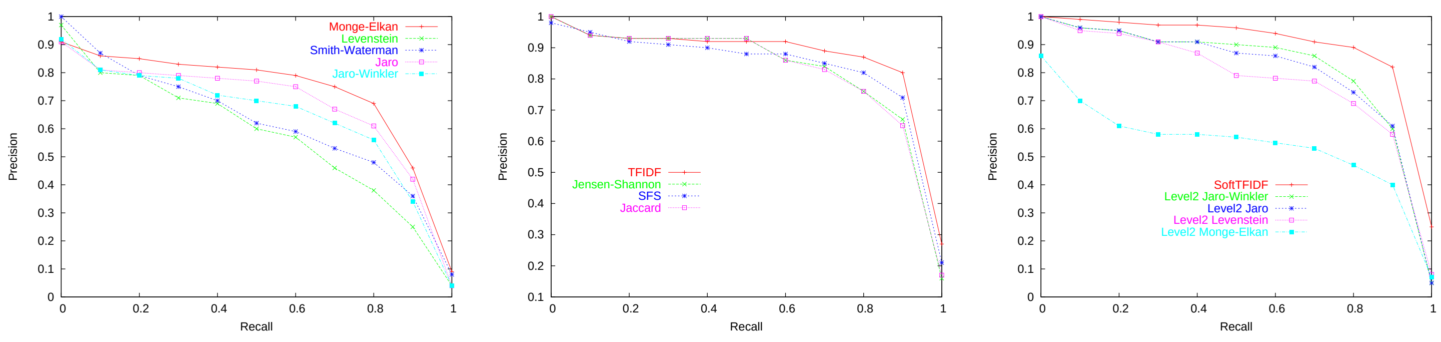

Figure 1: Relative performance of edit-distance measures compared to Monge-Elkan (left); token-based measures compared to TFIDF (middle); and hybrid measures compared to SoftTFIDF.

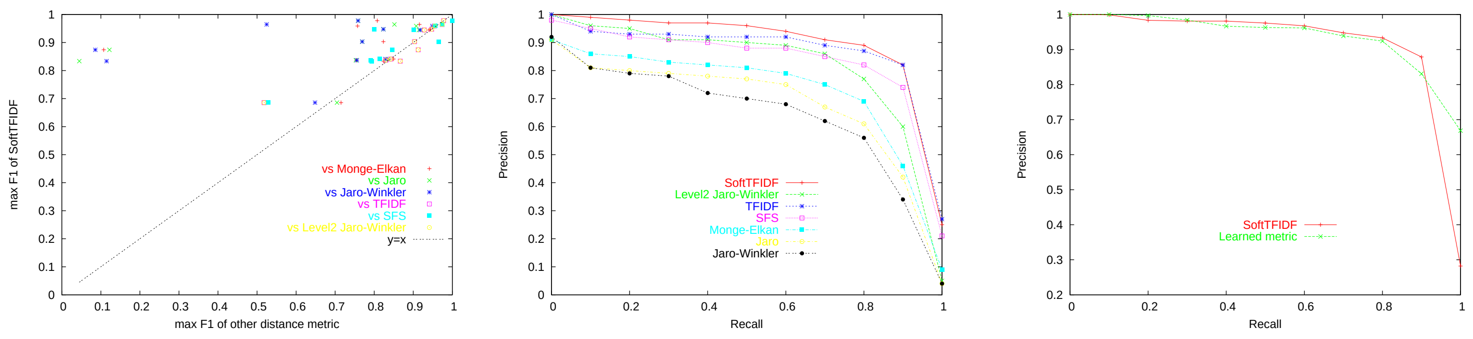

Figure 2: Relative performance of some "good" distance measures of each type on matching problems, relative to the SoftTFIDF metric, viewed as a scatter plot of maximum F1 values (left) and as 11-pt interpolated average precision (middle). SoftTFIDF compared to a learned combination of distance metrics (right).

Name	Src	#Strings	#Tokens
animal	1	5,709	30,006
bird1	1	377	1,977
bird2	1	982	4,905
bird3	1	38	188
bird4	1	719	4,618
business	1	2,139	10,526
game	1	911	5,060
park	1	654	3,425
fodorZagrat	2	863	10,846
ucdFolks	3	90	454
census	4	841	5,765

Table 1: Datasets used in experiments. Column 2 indicates the source of the data. Original sources are 1. (Cohen 2000) 2. (Tejada, Knoblock, & Minton 2001) 3. (Monge & Elkan 1996) 4. William Winkler (personal communication)

From Figure 1, we see that, on average, Monge-Elkan performs best of the edit-distance-like methods; that TFIDF performs best of the token-based methods; and that Soft-TFIDF performs best of the hybrid measures. Figure 2 provides some more detail on the performance of SoftTFIDF compared to the three best performing edit-distance like methods, the two best token-based methods, and the two best hybrid methods, using a similar methodology. Generally speaking, SoftTFIDF is the best overall distance measure for these datasets. In the scatter plot of Figure 2, each point is a dataset, positioned so that the max F1 score for SoftTFIDF is the x-axis position, and the max F1 score for some other method is the y-axis position; thus points above the line $y = x$ indicate better performance of SoftTFIDF.

Following previous researchers (Tejada, Knoblock, & Minton 2001; Cohen & Richman 2002; Bilenko & Mooney 2002) we also used a learning scheme to combine several of the distance functions above. Specifically, we represented pairs as feature vectors, using as features the numeric scores of Monge-Elkan, Jaro-Winkler, TFIDF, SFS, and SoftTFIDF. We then trained a binary SVM classifier (using SVM Light (Joachims 2002)) using these features, and used its confidence in the "match" class as a score. The results are summarized again in Figure 2 (using a three-fold cross-validation on nine of the matching problems). The learned combination generally slightly outperforms the individual metrics, including SoftTFIDF, particularly at extreme recall levels; however, it requires labeled training data, which the other metrics do not.

Record matching

Extending the architecture

For performance reasons, it is frequently crucial to avoid recomputing certain properties of a string—for instance, the tokenized form of the string. For this reason, SecondString interally manipulates objects called *string wrappers*, rather than strings. The *string wrapper* for string s allows additional information about s to be cached as needed.

Extending SecondString to handle record-matching is straightforward. We introduced a new type of string wrapper, called a *multi-string wrapper*. At creation time, this string wrapper splits a string s into subfields $s_1, \ldots, s_K$ according to a specified scheme (for instance, comma-separated fields). Subsequently any caller can access these pre-constructed fields of a string. In parallel, we introduced a *multi-string distance function*, which applies different distance functions to the different fields of a string, and then aggregates them.

Properties of the Dataset

We will now discuss some preliminary experiments in using SecondString on record (rather than string) matching. These experiments are preliminary in part because of lack of datasets with associated fields. Although the problem of record-linkage is well-motivated statistically, only two of the datasets used in our previous work have multiple fields—the Census and Cora datasets.

The Cora data (McCallum, Nigam, & Ungar 2000) was collected from postcript papers that were automatically converted to ascii text. The conversion procedure is error-prone, and introduced certain types of errors that are perhaps not representative of most text—for instance, a space is often mistakenly inserted after a capital "T". Tokens were grouped into fields by another automatic process, which again introduced errors of a particular type.

The Census data is artificially produced. In this case, care was taken to generate representative data. However, the Census dataset is generally speaking an outlier, compared to the other data sets in our test suite, with respect to relative performance of many of the string matching algorithms.

For example, the simple Levenstein distance metric performs worst, on average, of the distance metrics considered above. The difference between Levenstein and the stronger methods is also quite consistent. Compared to SoftTFIDF, for instance, Levenstein performs worse on 10 of the 11 datasets considered, and as shown by scatter plot of maximum F1 scores in Figure 3, the difference is often substantial. The one dataset on which Levenstein outperforms SoftTFIDF is the Census dataset.

It also appears that in Census, word frequency statistics do not appear to carry the same importance as they do in other settings. The weights used in TFIDF, and many other token-based distance metrics, assume that frequent words (like "Brown") are less important than infrequent ones (like "Zubinsky"). The relative performance of the token-based distance metrics suggests that, on Census, this assumption is incorrect[1]. On Census, TFIDF has a lower average precision precision than the Jaccard distance. Again, this result is an outlier; Jaccard only outperforms TFIDF on one other (smallish) dataset.

For these reasons, we decided to begin by exploring Census a little more carefully. We will focus our remarks on performance as measured by non-interpolated average precision, but other metrics behave similarly.

None of the token-based methods perform well on Census. Unusually, Jaccard performs about as well as any of the more complex methods. The hybrid methods perform somewhat better—but are still generally worse than the pure string-based methods. The best hybrid method for average precision is SoftTFIDF, and the best hybrid method for maximum F1 is Level 2 Jaro-Winkler (an version of Monge and Elkan's recursive matching scheme). Results for TFIDF, Jaccard, Level2 Jaro-Winkler, and SoftTFIDF are shown in Table 2.

The Jaro method performs well, with an average precision of 0.731, and a maximum F1 value slightly *higher* than the best hybrid method. This is surprising since it is does not seem to be intended for data of this type.[2] Motivated by this (and Jaro's usefulness in hybrid methods), we considered

[1]This is possibly because Census includes several households with a moderate number of individuals—for instance, there is a family of seven Mosqueras at one address, and a family of five Hoerrlings at another.

[2]Jaro seems designed for short strings, such as a last name, while Census contains contains first name, last name, middle initial, street number, and street name appended together.

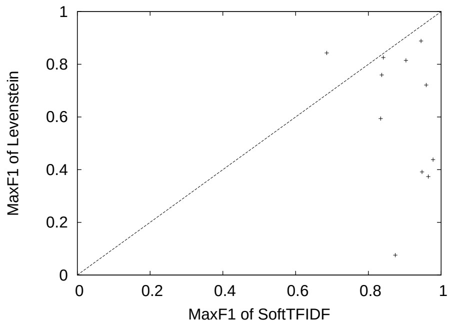

Figure 3: Max F1 score of Levenstein compared to the Max F1 score of SoftTFIDF

	MaxF1	AvgPrec	
SFS	0.528	0.357	
TFIDF	0.518	0.369	
Jaccard	0.567	0.402	
L2 Jaro-Winkler	0.746	0.770	
SoftTFIDF	0.685	0.782	
Jaro-Winkler	0.648	0.703	
Jaro	0.687	0.731	
NaiveAvgOverlap	0.697	0.731	
AvgOverlap	0.701	0.736	
Levenstein	0.832	0.901	
Jaro	0.728	0.789	trimmed
Scaled Levenstein	0.851	0.930	trimmed
Levenstein	0.865	0.925	trimmed

Table 2: Performance of various matching methods on Census

two variations of the Jaro method.

To understand them, we will first review the method itself. Consider the matrix M below, which compares the strings $s =$“WILLIAM” and $t =$“WILLLAIM”. The boxed entries are the main diagonal, and $M(i, j) = 1$ iff the i-th character of s equals the j-th character of t.

$$\begin{array}{c|ccccccc}
 & \text{W} & \text{I} & \text{L} & \text{L} & \text{I} & \text{A} & \text{M} \\
\hline
\text{W} & \boxed{\mathbf{1}} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 \\
\text{I} & \mathbf{0} & \boxed{\mathbf{1}} & \mathbf{0} & \mathbf{0} & 1 & 0 & 0 \\
\text{L} & \mathbf{0} & \mathbf{0} & \boxed{\mathbf{1}} & \mathbf{1} & \mathbf{0} & 0 & 0 \\
\text{L} & 0 & \mathbf{0} & \mathbf{1} & \boxed{\mathbf{1}} & \mathbf{0} & \mathbf{0} & 0 \\
\text{L} & 0 & 0 & \mathbf{1} & \mathbf{1} & \boxed{\mathbf{0}} & \mathbf{0} & \mathbf{0} \\
\text{A} & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \boxed{\mathbf{1}} & \mathbf{0} \\
\text{I} & 0 & 1 & 0 & 0 & \mathbf{1} & \mathbf{0} & \boxed{\mathbf{0}} \\
\text{M} & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1}
\end{array}$$

The Jaro metric is based on the degree to which strings s and t have characters “in common.” In terms of the matrix above, the i-th character of s is defined to be *in common with* t if $M_{i,j} = 1$ for some entry (i, j) that is “sufficiently close” to the main diagonal of M. In the Jaro metric, “sufficiently close” means that $|i - j| < \min(|s|, |t|)/2$ (shown in the matrix in bold.)

We looked at two variants of this rule. Let $M_{i_1,j_1}, \ldots, M_{i_n,j_n}$ be the matrix entries that contain a “1”, and let $d_k = i_k - j_k$. Notice that if s and t are highly similar, then most of the $M_{i,j}$ entries will be near the main diagonal, so most of d_k’s will be near zero; conversely, if the strings are dissimilar, the d_k’s will be widely scattered. The first Jaro variant we explored fits a mixture of two Gaussians to the d_k’s, where one “wide” Gaussian is constrained to have a high variance and zero mean, and the second’s variance and mean are unconstrained, and are set using E/M. The non-zero matrix entries M_{i_k,j_k} that have a high posterior probability of being generated by the “wide” Gaussian are considered to represent accidental matches, and the other matrix entries are considered to be related to the “common structure” of s and t. We thus measure overlap between s and t as

$$overlap(s, t) = \sum_{i=1}^{|s|} (1 - \prod_{j:M_{i,j}=1} p_{WIDE}(i - j))$$

		MaxF1	AvgPrec
SoftTFIDF	SVM	0.792	0.830
SoftTFIDF	AVG	0.803	0.810
Levenstein	SVM	0.890	0.928
Levenstein	AVG	0.870	0.920
Jaro	SVM	0.917	0.932
Jaro	AVG	0.897	0.922
Jaro-Winkler	SVM	0.930	0.933
Jaro-Winkler	AVG	0.915	0.900
Levenstein-Winkler	SVM	0.936	0.951
Levenstein-Winkler	AVG	0.912	0.916

Table 3: Performance of various structure-exploiting methods on Census

where $p_{WIDE}(d)$ is the posterior probability of generation by the “wide” Gaussian. Similarity is then the average of $overlap(s, t)$ and $overlap(t, s)$.

Performance of this method improves slightly over Jaro (shown in Table 2 under the name *AvgOverlap*). However, further experiments showed that replacing p_{WIDE} in the formula above with zero (shown as *NaiveAvgOverlap* in the Table) works nearly as well.

The best off-the-shelf method on Census is the Levenstein method. We conjectured that was because in our copy of Census, the fields (like last name, first name, street name, etc) are padded with blanks to be a uniform length. Intuitively, this simple “field alignment” should make comparisons easier with methods like Levenstein. However, this conjecture proved false: “trimming” the fields (by replacing multiple blanks with a single blank throughout) *improves* performance for Levenstein, as well as for Jaro.

Record-matching results

As a baseline for record matching, we consider first a very simple method which exploits record structure. Let $f(s, t)$ be a distance function on strings, and let $s = (s_1, \ldots, s_K)$ and $t = (t_1, \ldots, t_K)$ be a decomposition of s and t into fields. One extension of f to handle this field structure is to use the average distance between corresponding fields: $\frac{1}{K} \sum_i f(s_i, t_i)$.

Alternate rows of Table 3 (rows marked “AVG”) show performance of this baseline record-matching method.

As the second baseline extension to handle field structure, we adaptively combine the distances between the corresponding fields using a binary classifier. Specifically, we represented record-pairs as feature vectors, using as features the distances between corresponding fields. We then trained a binary SVM classifier (using SVM Light (Joachims 2002)) using these features, and used its confidence in the “match” class as a score. Rows marked “SVM” in Table 3 show performance of this record-matching method.

Using the field-structure in the above manner produces little improvement for Levenstein, but (not unexpectedly) substantially improves all the Jaro variants. With fields, the Jaro-Winkler also performs better than Jaro (without fields, Jaro-Winkler performs worse).

The Winkler variant is not immediately applicable to Levenstein method, since it requires a distance metric which is scaled between 0 and 1, whereas Levenstein counts the number of edit operations required to transform s to t. However, following the same technique used by Monge and Elkan (1996), the Levenstein distance also can be appropriately scaled. Applying the Winkler variant to scaled Levenstein gives the best average precision and maximum F1 score among the baseline methods.[3]

Concluding remarks

In previous work (Cohen, Ravikumar, & Fienberg 2003), we described an open-source Java toolkit of methods for matching names and records, and presented results obtained from

[3]Without fields, scaled Levenstein obtains a lower average precision but a higher max F1 than unscaled Levenstein (see Table 2).

using various string distance metrics on the task of matching entity names. These metrics include distance functions proposed by several different communities, including edit-distance metrics, fast heuristic string comparators, token-based distance metrics, and hybrid methods.

While string distances are often useful, they are not suitable for comparing entities with non-trivial structures. In many contexts, databases have non-trivial structure, and in many communities—notably, the statistical literature on probabilistic record linkage—pairs of entities are represented not by pairs of strings, but by vectors of "match features" such as names and categories for variables in survey databases. Motivated by this, we describe an extension to our toolkit which allows records to be compared.

To our knowledge, few publically available datasets which are naturally formed into records exist, which makes a comprehensive multi-technique, multi-technique comparison difficult. We thus focused on a single dataset, and presented a series of experiments with various baseline record-matching algorithms based on string comparisons between fields.

The best method described here is a scaled version of the Levenstein edit-distance metric, modified by a method proposed by Winkler for the Jaro distance metric, with the scores for corresponding fields being adaptively combined by a (SVM) binary classifier. This method can be implemented in a few lines of code using our toolkit, and it substantially improves over SoftTFIDF, the method which appears to work best on average on simple string comparisons in our previous paper. For instance, non-interpolated average precision is improved from 0.782 (for SoftTFIDF) to 0.951 (for the new method).

Further work will focus on collecting additional structured datasets and exploring other approaches to record matching, in order to improve this baseline performance.

Acknowledgements

The preparation of this paper was supported in part by National Science Foundation Grant No. EIA-0131884 to the National Institute of Statistical Sciences and by a contract from the Army Research Office to the Center for Computer and Communications Security with Carnegie Mellon University.

References

Belin, T. R., and Rubin, D. B. 1997. A method for calibrating false-match rates in record linkage. In *Record Linkage – 1997: Proceedings of an International Workshop and Exposition*, 81–94. U.S. Office of Management and Budget (Washington).

Bilenko, M., and Mooney, R. 2002. Learning to combine trained distance metrics for duplicate detection in databases. Technical Report Technical Report AI 02-296, Artificial Intelligence Lab, University of Texas at Austin. Available from http://www.cs.utexas.edu/users/ml/papers/marlin-tr-02.pdf.

Cohen, W. W., and Ravikumar, P. 2003. Secondstring: An open-source java toolkit of approximate string-matching techniques. Project web page, http://secondstring.sourceforge.net.

Cohen, W. W., and Richman, J. 2002. 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)*.

Cohen, W. W.; Ravikumar, P.; and Fienberg, S. E. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*. To appear.

Cohen, W. W. 2000. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems* 18(3):288–321.

Dagan, I.; Lee, L.; and Pereira, F. 1999. Similarity-based models of word cooccurrence probabilities. *Machine Learning* 34(1-3).

Durban, R.; Eddy, S. R.; Krogh, A.; and Mitchison, G. 1998. *Biological sequence analysis - Probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University Press.

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

Galhardas, H.; Florescu, D.; Shasha, D.; and Simon, E. 2000. An extensible framework for data cleaning. In *ICDE*, 312.

Hernandez, M., and Stolfo, S. 1995. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD*.

Jaro, M. A. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association* 84:414–420.

Jaro, M. A. 1995. Probabilistic linkage of large public health data files (disc: P687-689). *Statistics in Medicine* 14:491–498.

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

Larsen, M. 1999. Multiple imputation analysis of records linked using mixture models. In *Statistical Society of Canada Proceedings of the Survey Methods Section*, 65–71. Statistical Society of Canada (McGill University, Montreal).

McCallum, A.; Nigam, K.; and Ungar, L. H. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, 169–178.

Monge, A., and Elkan, C. 1996. The field-matching problem: algorithm and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*.

Monge, A., and Elkan, C. 1997. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery*.

Pasula, H.; Marthi, B.; Milch, B.; Russell, S.; and Shpitser, I. 2002. Identity uncertainty and citation matching. In *Advances in Neural Processing Systems 15*. Vancouver, British Columbia: MIT Press.

Raman, V., and Hellerstein, J. 2001. Potter's wheel: An interactive data cleaning system. In *The VLDB Journal*, 381–390.

Ristad, E. S., and Yianilos, P. N. 1998. Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(5):522–532.

Tejada, S.; Knoblock, C. A.; and Minton, S. 2001. Learning object identification rules for information integration. *Information Systems* 26(8):607–633.

Winkler, W. E. 1999. The state of record linkage and current research problems. Statistics of Income Division, Internal Revenue Service Publication R99/04. Available from http://www.census.gov/srd/www/byname.html.