Learning to Match and Cluster Large High-Dimensional Data Sets For Data Integration

William W. Cohen
WhizBang Labs
4616 Henry St.
Pittsburgh, PA 15213
william@wcohen.com

Jacob Richman
WhizBang Labs
4616 Henry St.
Pittsburgh, PA 15213
jsr@whizards.org

ABSTRACT

Part of the process of data integration is determining which sets of identifiers refer to the same real-world entities. In integrating databases found on the Web or obtained by using information extraction methods, it is often possible to solve this problem by exploiting similarities in the textual names used for objects in different databases. In this paper we describe techniques for clustering and matching identifier names that are both scalable and *adaptive*, in the sense that they can be trained to obtain better performance in a particular domain. An experimental evaluation on a number of sample datasets shows that the adaptive method sometimes performs much better than either of two non-adaptive baseline systems, and is nearly always competitive with the best baseline system.

Keywords

Learning, clustering, text mining, large datasets

1. INTRODUCTION

Data integration is the problem of combining information from multiple heterogeneous databases. One step of data integration is relating the primitive objects that appear in the different databases—specifically, determining which sets of identifiers refer to the same real-world entities. A number of recent research papers have addressed this problem by exploiting similarities in the textual names used for objects in different databases. (For example one might suspect that two objects from different databases named "USAMA FAYYAD" and "Usama M. Fayyad" " respectively might refer to the same person.) Integration techniques based on textual similarity are especially useful for databases found on the Web [1] or obtained by extracting information from text [6, 13, 11], where descriptive names generally exist but global object identifiers are rare.

Previous publications in using textual similarity for data integration have considered a number of related tasks. Although the terminology is not completely standardized, in this paper we define *entity-name matching* as the task of taking two lists of entity names from two different sources and determining which pairs of names are co-referent (*i.e.*, refer to the same real-world entity). We define *entity-name clustering* as the task of taking a single list of entity names and assigning entity names to clusters such that all names in a cluster are co-referent. Matching is important in attempting to join information across of pair of relations from different databases, and clustering is important in removing duplicates from a relation that has been drawn from the union of many different information sources. Previous work in this area includes work in distance functions for matching [14, 3, 9, 8] and scalable matching [2] and clustering [13] algorithms. Work in *record linkage* [15, 10, 21, 20, 7] is similar but does not rely as heavily on textual similarities.

In this paper we synthesize many of these ideas. We present techniques for entity-name matching and clustering that are scalable and *adaptive*, in the sense that accuracy can be improved by training.

2. LEARNING TO MATCH AND CLUSTER

2.1 Adaptive systems

We will begin defining the problems of adaptive matching and clustering by describing a very general notion of an adaptive system. Assume a source of *training examples*. Each training example is a pair $(x, y^{\square})$, where x is a *problem instance* and $y^{\square}$ is a *desired solution* to x. We will also assume a loss function, $Loss(y, y^{\square})$, measuring the quality of a proposed solution y relative to a desired solution $y^{\square}$. The goal of an adaptive system L is to take a set of training examples $(x_1, y_1^{\square}), \ldots, (x_m, y_m^{\square})$ and learn to propose "good" solutions to novel problems x_j. In other words, the input to L is the set $\{(x_i, y_i^{\square})\}_{i=1}^{m}$ and the output is a function f such that the loss $Loss(f(x_j), y_j^{\square})$ is small, where $y_j^{\square}$ is the desired solution for x_j. One simple, well-explored example of an adaptive system is classification learning.

2.2 Adaptive matching

Consider the task of learning to match names from some domain A with names from a second domain B. For exam-

SIGKDD '02 Edmonton, Alberta, Canada

ple, we might wish to learn to match a researcher's name and address with a university name if and only if the researcher is a□liated with that university. To formalize this, we let each problem instance x be a pair, $x = (A, B)$, where A and B are sets of strings. For instance, A might be names and addresses of researchers registered for KDD-02, and B might be names of universities in the United States. A solution y is a set of pairs $y = \{(a_1, b_1), \ldots, (a_k, b_k)\}$, specifically a subset of $A \times B$ that indicates which pairs are to be matched. A natural loss function $Loss(y, y^{\square})$ might be the size of the symmetric di□erence of y and $y^{\square}$: *i.e.* if $y = \{(a_i, b_i)\}_{i=1}^{k}$ and $y^{\square} = \{(a_j^{\square}, b_j^{\square})\}_{j=1}^{k^*}$ then

$$\begin{aligned} Loss(y, y*) &\equiv |\{(a_i, b_i) \in y : (a_i, b_i) \notin y^{\square}\}| \\ &+ |\{(a_i^{\square}, b_i^{\square}) \in y^{\square} : (a_i^{\square}, b_i^{\square}) \notin y\}| \end{aligned}$$

Other related measures are recall, precision, and F-measure—all of which are based on the symmetric di□erence of two sets.

Many matching problems are more constrained than this example. For instance, if the a'a and b's are entity names, and each $b \in B$ refers to a distinct entity, then it makes little sense for a proposed solution y to contain both (a, b) and $(a, b^{\prime})$. We define a *constrained adaptive matching problem* to be one in which the set of pairs in every desired pairing $y^{\square}$ is a one-to-one function.

Constrained matching problems are common—in fact, both of the matching problems considered in Section 4 are constrained. However, we consider here the more general case, which is useful (for instance) in matching datasets that may duplicates.

2.3 Adaptive clustering

The second problem we consider is *adaptive clustering*. In this case, each problem instance x is set of strings $D = d_1, \ldots, d_m$. A solution $y^{\square}$ is an assignment of the strings d_i to clusters, encoded as a function z from D to the integers between 1 and k (where k is the number of clusters).

For example, consider clustering descriptions consisting of a researcher's name, together with some additional piece of identifying information, such as his or her a□liation in July, 2002. A problem instance x would be a set of strings (like "William W. Cohen, Whizbang Labs", "W. Cohen, WhizBang Labs - Research", "Jude Shavlik, University of Wisconsin", etc) and a solution $y^{\square}$ would be a function z such that $z(d_1) = z(d_2)$ i□ d_1 and d_2 refer to the same person. Adaptive clustering is learning to cluster better given a sequence of training data in the form of (x, z) pairs.

3. SCALABLE ADAPTIVE METHODS

3.1 Clustering

The definitions above are extensions of the model for adaptive ranking systems described by Cohen, Singer and Schapire [5]. To oversimply slightly, Cohen, Singer and Schapire considered adaptive systems in which each problem instance x was an unordered set of objects $x = \{d_1, \ldots, d_m\}$, and each desired solution $y^{\square}$ was a total ordering over the objects in x. The problem of learning to order instances was addressed by learning a *preference function*, $p(d, d^{\prime})$—conceptually, a function $p : X \times X \rightarrow \{0, 1\}$ indicating if d should be ranked before $d^{\prime}$ in the desired ordering $y^{\square}$.

Adaptive matching and clustering can be implemented in an analogous way, by learning an appropriate *pairing function*. In the context of matching, a pairing function $h(a, b)$ is a binary function that indicates if a should be matched with b. In the context of clustering, $h(d, d^{\prime})$ indicates if d and $d^{\prime}$ should be placed in the same cluster. Figure 1 gives a simple algorithm for clustering using a pairing function.

To **train** from $\{(D_1, z_1), \ldots, (D_m, z_m)\}$:

1. Build a training sample S for the pairing function h.
   (a) Let $S = \emptyset$.
   (b) For $i = 1, \ldots, m$:
      i. Generate all pairs $(d, d^{\prime}) \in D_i \times D_i$.
      ii. Let $label(d, d^{\prime}) \equiv \begin{cases} + & \text{if } z_i(d) = z_i(d^{\prime}) \\ - & \text{otherwise} \end{cases}$
      iii. Add the labeled example $(d, d^{\prime})$ to S.
2. Train a classification learner on S. The result will be a hypothesis h that labels pairs $(d, d^{\prime})$ as positive or negative.

To **cluster** a new set $D = \{d_1, \ldots, d_n\}$:

1. Build a graph G with vertex set D, where an edge exists between d_i and d_j i□ $h(d_1, d_j) = +$.
2. Make each connected component of G be a cluster.

Figure 1: A naive clustering algorithm based on a learned pairing function

The algorithm of Figure 1 has two problems: a small number of errors in the learned pairing function h may lead to large mistakes in the clusters created; and the algorithm is ine□cient, since it requires generation of all pairs.

To address these problems, we modify Figure 1 in three ways. First, in training, we will enumerate only a limited number of "candidate" pairs in Step 1(b)i. Ideally the candidate set will be of manageable size, but will include all pairs $(d, d^{\prime})$ that should be clustered together.

Second, we will exploit the fact that classification learners can provide a *confidence* for their classifications. We replace Steps 1 and 2 with better methods for building and using the "pairing graph" G. In clustering Step 1, we construct the edges of G by using the same candidate-pair generation procedure used in training, and then weight each edge $(d, d^{\prime})$ by the *confidence* of the learned hypothesis h so that the label of $(d, d^{\prime})$ should be "+". In Step 2, we cluster the resulting edge-weighted graph (in this paper, using greedy agglomerative clustering). The resulting algorithm is shown in Figure 2.

We next consider the generation of candidate pairs (an operation often all called "blocking" in the record linkage literature). We use the *canopy method*, proposed by McCallum, Nigam and Unger [13]. This method relies on the ability to take an entity-name d and e□ciently find all nearby points $d^{\prime}$ according to some "approximate" distance metric. Following McCallum *et al* we used a TFIDF distance metric based on tokens. In this case, an inverted-index based ranked retrieval system can find nearby pairs quite quickly.

The canopy method, shown in Figure 3, begins with an empty set of candidate pairs, and operates by repeatedly

To **train** from $\{(D_1, z_1), \ldots, (D_m, z_m)\}$:

1. Build a training sample S for the pairing function h.
   (a) Let $S = \emptyset$.
   (b) For $i = 1, \ldots, m$:
      i. Let *CandidatePairs*(D) be a set of "candidate" pairings (d, d').
      ii. For each $(d, d') \in$ *CandidatePairs*(D), let
      $$\textit{label}(d, d') \equiv \begin{cases} + & \text{if } z_i(d) = z_i(d') \\ - & \text{otherwise} \end{cases}$$
      iii. Add the labeled example (d, d') to S.
2. Train a classification learner on S. The result will be a hypothesis h that labels pairs (d, d') as positive or negative.
3. Let $c(d, d')$ be the confidence given by h that the $h(d, d') = +$.

To **cluster** a new set $D = \{d_1, \ldots, d_n\}$ into K clusters:

1. Build a graph G with vertex set D, where an edge exists between d_i and d_j i $(d_i, d_j) \in$ *CandidatePairs*(D), and the weight of the edge between d and d_j is $c(d_i, d_j)$.
2. Perform greedy agglomerative clustering (GAC) on G to produce K clusters.
   (a) Create a singleton cluster to hold each vertex.
   (b) While there are more than K clusters:
      - Merge the two "closest" clusters, where cluster distance is the minimum distance between any members of the clusters.
3. Use the clustering produced by GAC on G as the clustering of D.

Figure 2: A better and more efficient adaptive clustering algorithm

picking a random "center point" d. After d is picked, all points d' that are "close enough" to d (within distance T_{loose}) are found. These "canopy" points are paired with each other, and the resulting pairs are added to the set of candidate pairs. Next, the set of possible "center points" is decreased by removing all points $\hat{d}$ within distance T_{tight} of d, where $T_{tight} < T_{loose}$. This process repeats until all possible center points are chosen.

For the benchmark problems considered in Section 4, it was fairly easy to find thresholds T_{tight} and T_{loose} that allow generation of nearly all "true" pairs (pairs that belong in a desired cluster) without generating too many spurious pairs.

In learning, two issues must be addressed: how to represent a pair (d, d'), and which learning algorithm to use. We explored several di erent classification learning systems, and di erent feature sets for representing pairs (d, d'). Here we will report results for a maximum entropy learner [16]. This learning system requires that examples be represented as a vector of binary features. Examples of the features used to encode a pair are shown in Table 1. Here the *edit distance* gives every character insertion and deletion unit cost, and *Jaccard distance* [18] is computed by treating d and d' as sets of tokens and using $|d \cap d'|/|d \cup d'|$ as a distance function.

In some of the test datasets we considered, the items to be clustered are not strings, but records consisting of several strings (for instance, a record containing a name and an address, or a bibliographic entry containing a title, author, date, and publication venue). For such datasets, a pair was encoded by extracting the features of Table 1 for every pair of fields, and combining all the features: for instance, in pairing name/address records, we computed the features $\text{SubstringMatch}_{\text{name}}$, $\text{SubstringMatch}_{\text{address}}$, $\text{PrefixMatch}_{\text{name}}$, $\text{PrefixMatch}_{\text{address}}$, ..., $\text{StrongNumberMatch}_{\text{name}}$, $\text{StrongNumberMatch}_{\text{address}}$.)

To compute *CandidatePairs*(D):

1. Let *CandidatePairs* = $\emptyset$.
2. Let *PossibleCenters* = D.
3. While *PossibleCenters* is not empty:
   (a) Pick a random d in *PossibleCenters*
   (b) Let *Canopy*(d) =
   $$\{(d, d') : d' \in D \wedge \textit{approxDist}(d, d') \leq T_{loose}\}$$
   In the implementation, *approxDist*(d, d') is based on TFIDF similarity, and *Canopy*(d) is computed e ciently using an inverted-index based retrieval method.
   (c) Add to *CandidatePairs* all pairs (d'_i, d'_j) such that both d'_i and d'_j are in *Canopy*(d).
   (d) Remove from *PossibleCenters* all points $\hat{d} \in D$ such that *approxDist*$(d, \hat{d}) \leq T_{tight}$
   (Again, $\{\hat{d} :$ *approxDist*$(\hat{d}, d) \leq T_{tight}\}$ can be computed quickly using inverted indices.)
4. Return *CandidatePairs*

Figure 3: Computing a set of candiate pairs using the canopy algorithm of McCallum, Nigam and Unger

3.2 Matching and Constrained Matching

It is fairly simple to adapt the algorithm above to the problem of constrained adaptive matching. Generation of candidate pairs is substantially easier, since one need only consider pairs (a, b) where $a \in A$ and $b \in B$. One possible technique is to use the canopy algorithm of Figure 3 with these modifications:

- in Step 2, let *PossibleCenters* = A;
- in Step 3b, let *Canopy*(a) = $\{(a, b) : b \in B$ and *approxDist*$(a, b) < T_{loose}\}$; and
- in Step 3d, let $T_{tight} = 0$ (*i.e.*, only remove a from the set of *PossibleCenters*).

A functionally equivalent but somewhat more e cient approach would be to use a soft join algorithm [3].

Learning a pairing function and construction of the graph G is identical. The greedy agglomerative clustering step,

SubstringMatch	true i one of the two strings is a substring of the other.
PrefixMatch	true i one of the strings is a prefix of the other.
EditDistance(k)	for $k \in \{0.5, 1, 2, 4, 8, 16, 32, 64\}$, true i the edit distance between the two strings is less than k.
MatchAToken(n)	true i the n-th token in d matches some token in d'.
MatchBToken(n)	analogous to MatchAToken(n).
MatchABigram(n)	like MatchAToken(n) but requires that both tokens n and n+1 match some token in d'.
JaccardDistance(k)	for $k \in \{0.1, 0.2, 0.4, 0.6, 0.8, 0.9\}$, true i the Jaccard distance between the sets of tokens in d and d' is less than k.
StrongNumberMatch	true if both d and d' contain the same number.

Table 1: Features used in learning the pairing function.

Benchmark	TFIDF Prec/Recall	Edit Distance Prec/Recall	Adaptive Prec/Recall
Cora	0.68/0.85 0.61/0.89	0.74/0.97	0.99/0.91 0.99/0.94
OrgName1	0.91/0.94 0.24/0.80	0.54/0.42 0.94/0.97	0.94/0.91 0.71/0.85
OrgName2	0.97/0.94 0.66/0.95	0.67/0.50 0.86/0.97	0.97/0.94 0.996/0.97
Restaurant	0.98/0.98 0.67/0.97	0.83/0.83 0.87/0.87	1.00/1.00 0.95/0.95
Parks	0.98/0.98 0.97/0.97	0.97/0.97 0.97/0.97	0.98/0.98 0.97/0.97

Table 3: Experimental results: precision and recall

Benchmark	TFIDF	Edit Distance	Adaptive
Cora	0.751 0.721	0.839	**0.945** **0.964**
OrgName1	**0.925** 0.366	0.633 **0.950**	0.923 0.776
OrgName2	**0.958** 0.778	0.571 0.912	**0.958** **0.984**
Restaurant	0.981 **0.967**	0.827 0.867	**1.000** 0.950
Parks	0.976 **0.967**	0.967 **0.967**	**0.984** **0.967**

Table 4: Experimental results: F-measure

however, should be replaced with an operation that enforces the constraints required for constrained adaptive matching. This can be done by computing the minimal weight cutset of G, and returning the edges of this cutset as the pairing. We have experimented with both a greedy approach and an exact minimization (which exploits the fact that the graph is bipartite [17]). The experiments in this paper are for a simple greedy mincut-finding algorithm, which is more e cient for large graphs.

3.3 Relationships

We note that the problems of learning pairing functions, clustering, and matching are closely related, but distinct. In unconstrained matching, the pairs do not correspond immediately to clusters, since pairs may overlap, but clusters are disjoint. In constrained matching, matching can be reduced to clustering, but exploiting the additional constraint that a pairing is one-to-one can substantially change the di culty of a clustering task. Finally, while learning a pairing function is a natural way of making a clustering system adaptive, obtaining an accurate hypothesis h does not mean that the ensuing clustering will be any good, as it is possible for small errors in h to cause large clustering errors [4].

4. EXPERIMENTS

We used several datasets for evaluation purposes. Two of the datasets require clustering, and two require matching. The first clustering dataset, `Cora`, is a collection of paper citations from the Cora project [12, 13]. The second dataset, `OrgName`, is a collection of 116 organization names. We considered two target clusterings of this data, one into 56 clusters, and one into 60 clusters.[1]

There are also two constrained matching datasets. The `Restaurant` dataset contains 533 restaurants from one restaurant guide to be matched with 331 from a second guide.[2] The `Parks` dataset contains 388 national park names from one listing and 258 from a second listing, with 241 names in common.

We assumed that the number of intended clusters K is known. For `OrgName`, `Restaurant`, and `Parks`, we constrained all systems (adaptive and non-adaptive) to produce the true number of clusters or pairings. For `Cora`, we wished to compare to the best previous clustering result, which was obtained varying cluster size widely. We tried two di erent target cluster sizes and report the one which gave the best result, obtained setting K to 1.5 times the true number of clusters.

To evaluate performance we split the data into two partitions, then trained on the first and tested on the second, and finally trained on the second and tested on the first. The datasets used are summarized in Table 2. For each dataset, we record the number of entities in each partition; the number of desired clusters or pairs; the thresholds used for the canopy algorithm; and the number of positive and negative examples generated.

As success measures for the algorithms, we used several di erent definitions of "loss". Recall that for matching, a solution y is a set of pairs (a, b). Following the usual con-

[1] The di erence is that in the second clustering, di erent branches of an organization (such as "Virginia Polytechic Institute, Blacksburg" and "Virginia Polytechnic Institute, Charlottesville") are considered distinct, and in the first, they are not. Thanks to Nick Kushmeric for providing this data.

[2] Thanks to Sheila Tejada for providing this data.

Benchmark Name	Cluster or Match?	Partition Size		Thresholds		Pairing Examples		Potential
		# Entities	# Clusters	T_{tight}	T_{loose}	# Pos	# Neg	Recall
Cora	(c)	991	65	0.36	0.53	19,111	7,379	0.972
		925	64			15,431	8,711	0.998
OrgName1	(c)	60	42	0.24	0.40	33	56	1.000
		56	17			196	250	1.000
OrgName2	(c)	53	34	0.24	0.40	36	48	1.000
		63	22			270	181	1.000
Restaurant	(m)	430	52	0.28	0.93	52	426	1.000
		434	60			59	153	0.983
Park names	(m)	325	124	0.30	0.90	124	304	0.992
		321	117			117	357	0.975

Table 2: Datasets used in the experimental evaluation

vention in information retrieval, we define the *recall* of y relative to $y^¤$ to be $|y \cap y^¤|/|y^¤|$, the *precision* of y relative to $y^¤$ to be $|y \cap y^¤|/|y|$, the *F-measure* of y relative to $y^¤$ to be the harmonic mean of recall and precision.[3]

For clustering algorithms, recall that a problem instance x is a set of objects D, and a solution $y^¤$ is a mapping z from D into the integers $\{1, \ldots, K\}$, and define *pairs*(D, z) to be the set of all pairs $\{(d, d') \in D \times D : z(d) = z(d')\}$. We will define recall and precision in terms of *pairs*(D, z): *i.e.*, we define the *recall* of z relative to $z^¤$ is $|$*pairs*$(D, z) \cap$ *pairs*$(D, z^¤)|/|$*pairs*$(D, z^¤)|$, and the *precision* of z relative to $z^¤$ is $|$*pairs*$(D, z) \cap$ *pairs*$(D, z^¤)|/|$*pairs*$(D, z)|$. The final column of Table 2 shows the maximum recall obtainable using the *CandidatePairs* produced by the canopy algorithm.[4]

In addition to the algorithm described in Section 3, we considered two additional clustering/matching algorithms as performance baselines. The first one replaces $c(a, b)$ in the graphs above with Levenstein edit distance. Applied to clustering, this aseline algorithm is similar to the algorithm proposed by McCallum, Nigam and Unger; applied to matching, it is similar to the method proposed by Monge and Elkan[14]. The second baseline replaces $c(a, b)$ with TFIDF distance, using the formula given in [18], which is similar to the algorithm used in WHIRL [2].

The experimental results for these algorithms on the datasets of Table 2 are shown in Tables 3 and 4. The baseline results for edit distance are taken from [13], who used hand-tuned edit distance, and unlike the other entries in the table, they apply to the whole set, rather than a single partition. In Table 4, the best F-measure obtained on each problem is placed in bold.

A first observation on the results of Table 4 is that neither baseline system appears to outperform the other. Discounting `Cora` (for which the edit-distance function was hand-engineered), the TFIDF-based baseline obtains a better F1 score than the distance-function baseline on five runs, performs worse on two runs, and performs identically on one run. This confirms our belief that both TFIDF and edit-distance distance metrics are useful in data integration settings.

The adaptive method does far better than either baseline technique on the `Cora` dataset. Notice that the `Cora` dataset is the largest of the datasets considered, as well as the one for which the baseline methods perform the worst; hence it o□ers the most opportunity for adaptive techniques to improve performance. In the remaining eight runs, the adaptive technique performs best on five, and nearly equals the best result on two more (the first split of `OrgName1` and the second split of `Restaurant`). Thus on nine of the ten partitions, the adaptive method obtains results comparable to or better than the best of the baseline approaches.

The adaptive methods performs poorly on only one of the ten runs—the second partition of `OrgName1`. We conjecture that for this dataset (by far the smallest we considered) the constraints on partitioning used above resulted in substantial variation across the two partitions used for training and testing.[5]

5. CONCLUSIONS

We have presented a scalable adaptive scheme for clustering or matching entity names. Experimental results with the method are comparable to or better than results obtained by clustering or matching with two plausible fixed distance metrics.

As noted above, our formalization of adaptive clustering and matching is inspired by the model of "learning to order" of Cohen, Schapire, and Singer [5]. They consider adaptive ordering systems and show that this problem can be solved by supervised learning of a binary ordering relation, followed by a greedy method for constructing a total order given a set of (possibly inconsistent) binary ordering decisions. They also give provable bounds on the loss of such

[3] That is, $F = \frac{2¢P¢R}{(P+R)}$.

[4] Creating appropriate partitions for training and test is non-trivial, since one must ensure that the test cases are independent of the training cases, and a simple random partition of would likely lead to a situation in which some of the intended clusters were split between the training and test sets. To avoid this, we split the data so that no algorithm that considers only pairs produced by the canopy algorithm would ever consider a pair containing one instance from the test set and one instance from the training set. A disadvantage of this procedure is that it was sometimes impossible to create well-balanced splits, biasing the results away from adaptive methods.

[5] Notice that the TFIDF-based baseline system does much better than the edit-distance based baseline on the first partition, but that the opposite holds on the second partition. Thus even the trivial adaptive system that chooses the better of the two baseline systems based on training data would perform poorly. The size of the pairing-function training sets and the number of entities per cluster is also varies greatly in the two partitions.

a system. Finding such bounds for adaptive clustering or learning remains a problem for future work.

The architecture of the adaptive matching and clustering method is modeled after the system of McCallum, Nigam and Unger [13]. However, in our system, we consider matching as well as clustering, we also replace a fixed, hand-coded, edit-distance metric with a learned pairing function. Our focus on general-purpose adaptive clustering and matching methods also distinguishes this work from previous work on general-purpose non-adaptive similarity metrics for entity names (e.g. [9, 14]) or general frameworks for manually implementing similarity metrics (e.g.,[8]).

The "core" idea of learning distance functions for entity pairs is not new—there is a substantial literature on the "record-linkage" problem in statistics (e.g., [10, 20] much of which based on a record-linkage theory proposed by Felligi and Sunter [7]. The maximum entropy learning approach we use has an advantage over Felligi-Sunter in that it does not require features to be independent, allowing a broader range of potential similarity features to be used; at the same time the method is fairly e□cient, in contrast to Felligi-Sunter extensions based on latent class models [19].

ChoiceMaker.com, a recent start-up company, has also implemented a matching procedure based on a maximum entropy learner. We extend this work with a systematic experimental evaluation, use of canopies to eliminate the potentially quadratic cost of learning and clustering, and application of the pairing function to both clustering and matching.

A number of enhancements to the current method are possible. In future work we hope to examine other features; for instance, one notable current omission is the lack of any feature that directly measures TFIDF similarity. We also hope to compare these methods directly to other matching techniques developed in the statistical literature [19, 20].

6. REFERENCES

[1] William W. Cohen. Reasoning about textual similarity in information access. *Autonomous Agents and Multi-Agent Systems*, pages 65–86, 1999.

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

[3] William W. Cohen. WHIRL: A word-based information representation language. *Artificial Intelligence*, 118:163–196, 2000.

[4] William W. Cohen and Jacob Richman. Learning to match and cluster entity names. In *Proceedings of the ACM SIGIR-2001 Workshop on Mathematical/Formal Methods in Information Retrieval*, New Orleans, LA, 2001.

[5] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[6] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

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

[8] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: an extensible data-cleaning tool. In *Proceedings of ACM SIGMOD-2000*, June 2000.

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

[10] B. Kilss and W. Alvey. Record linkage techniques—1985. Statistics of Income Division, Internal Revenue Service Publication 1299-2-96. Available from http://www.bts.gov/fcsm/methodology/, 1985.

[11] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[12] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

[13] A. McCallum, K. Nigam, and L. Ungar. E□cient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

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

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

[16] Kamal Nigam, John La□erty, and Andrew McCallum. Using maximum entropy for text classification. In *Proceedings of Machine Learning for Information Filtering Workshop, IJCAI '99*, Stockholm, Sweden, 1999.

[17] H.A. Baier Saip and C.L. Lucchesi. Matching algorithms for bipartite graph. Technical Report DCC-03/93, Departamento de Cincia da Computao, Universidade Estudal de Campinas, 1993.

[18] Gerard Salton, editor. *Automatic Text Processing*. Addison Welsley, Reading, Massachusetts, 1989.

[19] W. E. Winkler. Improved decision rules in the Felligi-Sunter model of record linkage. Statistics of Income Division, Internal Revenue Service Publication RR93/12. Available from http://www.census.gov/srd/www/byname.html, 1993.

[20] W. E. Winkler. 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, 1999.

[21] William E. Winkler. Matching and record linkage. In *Business Survey methods*. Wiley, 1995.

Acknowledgments

The authors thank Andrew McCallum for numerous helpful suggestions.