Learning to Match and Cluster Entity Names

William Cohen* wcohen@wizbang.com

Jacob Richman*Φ jrichman@whizbang.com

*WhizBang Labs
4616 Henry St,
Pittsburgh, PA 15213

ΦSchool of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

1 Introduction

Information retrieval is, in large part, the study of methods for assessing the similarity of pairs of documents. Document similarity metrics have been used for many tasks including *ad hoc* document retrieval, text classification [YC1994], and summarization [GC1998,SSMB1997]. Another problem area in which similarity metrics are central is *record linkage* (e.g., [KA1985]), where one wishes to determine if two database records taken from different source databases refer to the same entity. For instance, one might wish to determine if two database records from two different hospitals, each containing a patient's name, address, and insurance information, refer to the same person; as another example, one might wish to determine if two bibliography records, each containing a paper title, list of authors, and journal name, refer to the same publication.

In both of these examples (and in many other practical cases) most of the record fields contain text---person names, paper titles and so on. It is natural to ask if document similarity metrics developed in the IR community can be useful for matching and clustering entity names. Recently there has been a substantial amount of work in addressing this question. Somewhat surprisingly, traditional IR document similarity metrics have experimentally been useful in comparing the very short "documents" that are used as informal names for entities. TF-IDF similarity metrics have been shown to be competitive with or superior to hand-coded similarity metrics and string edit-distance based metrics in a variety of domains, ranging from bibliographic references for scientific papers [LGB1999] to names of animal species and popular movies [C2000]. Even when TF-IDF similarity is not the most accurate similarity metric, it is often the fastest reasonable metric, and can be usefully applied in combination with other, more computationally expensive similarity tests [MNRS2000].

In this paper, we propose and describe several techniques for adaptively modifying document similarity metrics. We will evaluate these techniques but on entity-matching and entity-clustering tasks---in particular, we evaluate using learned similarity metrics in combination with simple TF-IDF based distance schemes. Our hope is that learned distance metrics can improve on general-purpose, task-independent distance metrics like TF-IDF, and ultimately provide performance comparable to engineered, domain-specific distance metrics and/or entity matching schemes (e.g. [HS1998, GFS2000]). Although our evaluation is solely in the context of entity-name matching and clustering, we conjecture that similar learning methods will be useful in other tasks involving document similarity metrics, such as document retrieval, summarization, classification, and clustering.

Below we more precisely define the problem, present an algorithm, present our experimental results. We then discuss related work and present some concluding remarks.

2 Adaptive matching and clustering methods

We begin by formally defining the problems of adaptive matching and clustering.

Formal preliminaries. In a general learning setting, we assume a source of *training example* pairs *(x,y)* where *x* is a problem instance and *y* is a solution, and also a *loss function, Loss(y,y*)*

measuring quality of the proposed solution y relative to some optimal or intended solution y^*. The prototypical example is binary classification learning, in which each y is taken from the set $Y=\{0,1\}$, each x is a binary vector from $X=\{0,1\}^n$, and $Loss(y,y^*)=1$ if y and y^* are different *and* $Loss(y,y^*)=0$ if y and y^* are the same.

The object of a learning system L is to take a set of training examples $(x_1,y_1),\ldots,(x_m,y_m)$ and learn to propose "good" solutions to novel problems x_j. There are many ways of formalizing this goal precisely. Our goal in this paper is not to prove deep theorems about adaptation but simply to motivate and precisely describe a particular class of adaptive systems; thus, we will adopt a very simple notion of "goodness". Imagine a repeated game where at each time step i the learner first receives a problem instance x_i, then proposes a solution y_i, and finally sees the desired solution y_i^*. After each step the learner suffers a penalty of $Loss(y_i,y_i^*)$. We will say that a system *converges after T steps* if, for all $N>T$ and all sequences $(x_1,y_1),\ldots,(x_N, y_N)$, the cumulative value of $Loss(y_i,y_i^*)$ for $i>T$ is zero.

Adaptive matching. In this paper we will consider adaptive systems where the problem instances x and solutions y are somewhat more complex than is the case in supervised learning. In *adaptive matching*, we formalize the task of learning to match names from some domain A with names from a second domain B. For example, we might wish to learn to match a researcher's name and address with a university name exactly when the researcher is affiliated 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 SIGIR-01, and B might be names of universities. A solution y is a set of pairs $y=\{(a_i,b_j)\}$, specifically a subset of the cross-product of A and B, which indicates which pairs are to be matched. A natural loss function $Loss(y,y^*)$ might be the size of the symmetric difference of y and y^*. In the experiments of this paper, we will also consider measures like recall, precision, and F-measure of y relative to y^*.

Many matching problems are more constrained than the one suggested here. Often it is the case that each a is intended to match b only when a and b refer to the same entity: an example of this might be matching names and addresses of SIGIR researchers with entries in a phone book. If the a's and b's are entity names, and each name b refers to a distinct entity, then it makes little sense for a proposed solution y to contain both (a,b) and (a,b'). In this paper we will consider also the *constrained adaptive matching problem*, in which the proposed pairing y is required to be one-to-one.

Adaptive clustering. The second problem we consider is *adaptive clustering*. In this case, x is a set of strings $D=\{d_i\}$, and y is an intended assignment of these strings to clusters, encoded as a function $z:D\rightarrow\{1,\ldots,k\}$, where k is an integer. There are many candidates for a loss function $Loss(y,y^*)$. One obvious constraint is that the loss function should not depend on the cluster names—i.e., that permuting the values of $z(d_j)$ shouldn't change the loss of z relative to an intended assignment. One convenient choice is to define $Loss(z,z^*)$ as the number of pairs (d,d') from D such that z and z^* disagree as to whether d and d' should be placed in the same cluster. Stated another way, let $pairs(z,D)=\{(d,d'):d\neq d' \text{ and } z(d)=z(d')\}$. One can then define $Loss(z,z^*)$ to be the size of the symmetric difference between $pairs(z,D)$ and $pairs(z^*,D)$—or any related measure, e.g., one based on recall, precision, or F-measure. This definition of loss is closely related to the cost function used in all-pairs clustering (e.g., [BH1994,GWW1998]).

A natural variation of this problem is when k, the number of clusters, is known. In this case we simply let the problem instance be a pair (D,k). We call this variation *adaptive k-clustering*.

3 Relationships between matching, clustering, and learning

Similarities. As we have formulated the problem, there are close connections between adaptive clustering, adaptive matching, and classification learning. For instance, there are some simple, natural reductions between these problems.

Observation 1: Constrained adaptive matching can be reduced to clustering. Specifically, given an adaptive clustering algorithm LC that converges after T steps, we can construct a constrained adaptive matching algorithm LM that converges after T steps. Since every desired match y^* is one-to-one, there is one-to-one function $g(a)$ such that each (a,b) in y^* can be written as $(a,g(a))$. We will assume a numbering of the elements in B, so that $j(b)$ is an integer from 1 to |B|. To construct the matching algorithm, we convert each matching instance ((A,B),y) to a clustering instance (D,z) as follows:

- D is the union of A and B;
- For d in B such that no pair (a,b) is in y, let $z(d)$=|B|+1;
- For d in B such that some pair (a,b) is in y, let $z(d)=j(d)$;
- For d in A such that no pair (a,b) is in y, let $z(d)$=|B|+1;
- For d in A such that $(a,g(a))$ is in y, let $z(d)=j(g(a))$.

Intuitively, each pair (a,b) is a cluster named by its "b" element, and every element a and b is mapped by z either to this cluster, or to a cluster of unmatched elements (numbered |B|+1). Clearly, after training LC, one can use it to produce a clustering z for a given new instance (A,B), and from z one can reconstruct a pairing y.

It should be noted *unconstrained* matching cannot be reduced to clustering by the same procedure, however: absent constraints, pairs can overlap in arbitrary ways, while clusters must be pairwise disjoint.

Observation 2. Adaptive clustering can be reduced to adaptive matching. Specifically, given an adaptive clustering algorithm LM, that converges in T steps, we can construct a constrained adaptive matching algorithm LC that converges in T steps. The idea behind this reduction is to pair objects together iff they belong in the same cluster. In more detail, each clustering instance ((k,D),z) is converted to a matching instance where A=B=D and $y=pairs(z,D)$. These examples are used to train LC. After training, a clustering problem instance D is converted again to a matching problem instance with A=B=D, and a cluster is derived from the proposed set of pairs y. If we assume y has zero loss, then finding the associated cluster is trivial; for instance, one could create a cluster for each connected component of the set of pairs.

Finally, we note that there is a close relationship between adaptive matching and binary classification learning. To make this clear, let us first introduce the idea of *staged classification learning,* where a learner is forced in each step to make classification predictions on a set of objects, rather than a single object. More precisely, consider an adaptive system where each problem instance is a vector of objects $\mathbf{x}=\langle x_1,\ldots,x_k\rangle$, each solution is a vector of bits $\mathbf{y}=\langle y_1,\ldots y_k\rangle$, and *Loss*(**y**,**y***) is just the Hamming distance between **y** and **y***. It is easy to see that adaptive matching can be reduced to this minor variation on classification learning:

Observation 3. Adaptive matching can be reduced to staged classification learning. To reduce adaptive matching to staged classification learning, we convert each matching instance ((A,B),y) to a pair (**x**,**y**) as follows:

- **x** is a vector containing all pairs (a,b);
- For each pair (a,b) in **x**, the corresponding component of **y** is 1 if (a,b) is a proposed pair in y and 0 otherwise.

Together with Observation 2, this suggests that adaptive matching and clustering could be implemented by using supervised learning to learn a "pairing function"—a function mapping each pair (a,b) to 1 or 0, indicating if a and b should be matched together (or placed in the same cluster.)

Finally, we note that (ordinary) supervised learning can also be reduced to adaptive matching:

Observation 4. Supervised learning can be reduced to adaptive matching. To reduce adaptive matching to staged classification learning, we convert each learning instance *(x,class(x))* to a matching problem ((A,B),*y),* where A={*x*}, B={0,1}, and *y* contains the single pair *(x,class(x))*.

Differences. These simple and natural reductions show that the three problems of learning, constrained adaptive matching, and adaptive clustering are in some sense equivalent: in particular, a perfect (zero-loss) solution to one problem is a perfect solution to all.

However, zero-loss solutions are unlikely to be achievable in practice. Closer examination shows that non-zero losses can be enlarged dramatically or decreased dramatically if these reductions are used. For instance, consider an adaptive matching system that is derived from an adaptive clustering system, as in Observation 1, and imagine a case where A={$a_1,...,a_n$}, B={b}, and the intended pairing y^* is empty. Suppose the clustering algorithm places every a_i and b in the same cluster, i.e. $z(d)$ is a constant function. The associated pairing for this cluster is $\{(a_1,b),...,(a_n,b)\}$ which has a loss of n; however, the loss of the cluster is $|pairs(z,D)|$ which is $(n+1)*n$.

By the same token, consider an adaptive clustering system derived from an adaptive matching system, as in Observation 2, and consider a problem instance where D={$d_1,...,d_n$}, the intended clustering is to place every item in its own cluster (i.e., $z(d_i)=i$), and the pairing proposed by the matching system is $\{(d_1,d_2), (d_2,d_3),..., (d_{n-1},d_n)\}$. We did not specify how to construct a cluster from a pairing in the remarks above, but notice that if the pairing has zero loss, it must correspond to a "correct" clustering, and the mapping is trivial. Creating a cluster for each connected component of the pairing graph will lead to a cluster in which $pairs(D,z)$ contains all $n*(n-1)$ pairs, for a loss of $n*(n-1)$, whereas the loss of the matching system is again only n.

The only reductions above that are tight with respect to loss are the ones associated with Observations 3 and 4, between adaptive matching and supervised learning. Even here additional loss might be suffered between a constrained adaptive matching system and a supervised learning system.

Remarks. To summarize, we have noted that the problems of adaptive matching and clustering are superficially similar to each other, and also to supervised learning. However, these similarities are not strong enough to show that a good solution to one problem will *necessarily* be a good solution to the other; nor are the obvious similarities to supervised learning strong enough to show that a good supervised learning system will *necessarily* perform well when used for adaptive clustering, or constrained adaptive matching.

In the remainder of the paper we will experimentally evaluate adaptive matching and clustering systems that are based on using supervised learning to classify pairs of entity names.

4 A Proposed Algorithm for Adaptive Matching and Clustering

Observations 2 and 4 suggest that both adaptive matching and clustering can be implemented by learning an appropriate "pairing function": a function $h(a,b)$ which predicts if a should be matched or clustered with b. To make this approach practical, several issues need to be addressed. One issue is that the cost of evaluating or learning a pairing function h is quadratic in |D|, which is often impractically large.

Many techniques are known for "blocking", i.e., finding a subset of pairs which are likely to be matched together. We will use the "canopy" method proposed by McCallum, Nigam and Unger [MNU2000]. The canopy assumes an easy-to-compute distance metric $dist(d,d')$ and two thresholds T^{tight} and T^{loose}, where $T^{loose} \geq T^{tight}$. It can be summarized as follows:

1. Let PossiblePairs be the empty set.
2. Let PossibleCenters be the set of objects D for which h(d,d') should be computed.
3. While PossibleCenters is not empty:
    a. Pick a random *d* in PossibleCenters.
    b. Add all pairs *(d,d')* such that *dist(d,d')*<T^{loose} to PossiblePairs.
    c. Remove all points d'' such that *dist(d,d'')*<T^{tight} from PossibleCenters.

The output of this procedure is the set PossiblePairs. In learning, we will train only on pairs from this set, and in evaluation, we will assume that *h(d,d')*=0 for every pair not in this set. Following [MNU2000] we used for *dist(d,d')* cosine distance on a tokenized version of the entity name. Thus finding the sets of points *d*' and *d''* in steps 3a and 3b can be done quickly with a ranked retrieval search.

So far we have ignored the fact that most practical learners do not give a 0/1 prediction for *h(a,b)*): instead they will give a numerical confidence, perhaps interpretable as a probability that *a* should be matched with *b*. Below we will use *c(a,b)* to denote this confidence. Another issue that must be addressed is that the pairs accepted by the learned pairing function need not obey all necessary constraints, e.g., the constraints associated with constrained adaptive matching, or the constraint on cluster size in *k*-clustering.

To enforce the constraints for constrained adaptive matching, we build a graph G where the vertices are entity names from A and B and the edges are weighted by the confidence of the classification learner, *c(a,b)*, and then compute the minimal weight cutset of this graph. We have experimented with both a greedy approach and an exact minimization (which exploits the fact that the graph is bipartite [SL1993]). The experiments in this paper are for the simple greedy approach.

To enforce the cluster size constraint, we build an analogous graph over the vertex set D and perform greedy agglomerative clustering (GAC). Our implementation of GAC follows [MNU2000]. Initially, a singleton cluster is created to hold each element. One then repeatedly merges together the "closest" clusters (where distance between clusters is defined is the minimum distance between cluster members) until only *k* clusters remain.

To summarize, the proposed algorithm is the following.

To train from a set of clustered (or matched) datasets:

1. Run the canopies algorithm (above) on each dataset to generate a set of PossiblePairs.
2. For each pair *(a,b)* in some set of PossiblePairs, label *(a,b)* as positive iff *a* and *b* are in the same cluster (respectively, are matched together). Add the labeled pairs to a training set *T*.
3. Train a classifier on the training set *T* and obtain a classifier *h*, and *c(a,b)* denote the confidence of this classifier on the pair *(a,b)*.

To cluster (or match) a dataset *D* using the result of training:

1. Run the canopies algorithm to obtain a set of PossiblePairs.
2. For each pair *(a,b)* in the set of PossiblePairs, compute *c(a,b),* creating a graph with vertex set *D* and edge weights *c(a,b)*.
3. Perform greedy agglomerative clustering on the graph (respectively, find the minimal weight cutset) and return the result.

5 Experimental Results

We explored several different instantiations of the adaptive matching and clustering algorithm proposed above. We used several different classification learning systems, and several different feature sets for representing pairs (*a,b*). Here we report results for a maximum entropy learner [DLR77]. Some of the features used to encode a pair are shown in Figure 1, below. For datasets

with multiple fields such as bibliographic references or name address pairs, the features are extracted for every field.

We considered two baselines for performance. The first one replaces *c(a,b)* in the graphs above with a fixed string edit distance. (Specifically, we compute the edit distance using uniform costs for any character insertion or deletion. For clustering, this is similar to the algorithm proposed by McCallum, Nigam and Unger.) The second baseline replaces *c(a,b)* with TF-IDF distance, using the formula given in [S1989].

SubstringMatch: true iff one of the two strings is a substring of the other.
PrefixMatch: true iff one of the strings is a prefix of the other.
EditDistanceK: for k=0.5,1, 2,4,8,16,32,64, the feature EditDistanceK is true iff the edit distance between the two strings is less than k.
MatchATokenN: true iff the N-th token in *a* (from the beginning) matches some token in *b*. This feature is computed for every possible value of N. MatchBTokenN is analogous.
MatchABigramN: like MatchATokenN but requires that both tokens N and N+1 match some token in *b*. Again, this is computed for all possible values of N, and MatchBBigramN is analogous.
JaccardDistanceK: true iff the Jaccard distance between the sets of tokens in *a* and *b* is less than K. (The Jaccard distance between sets S and T is $|S \cup T| / |S \cap T|$). This is computed for the thresholds K=0.1,0.2,0.4,0.6,0.8,0.9, and 1.0.
StrongNumberMatch: true if both *a* and *b* contain the same number.

Figure 1. Some features used in learning the pairing function

We also used several datasets for evaluation purposes. These are summarized in Table1. The first clustering dataset, *Cora*, is a collection of paper citations from the Cora project [NMU2000]. The second dataset, *OrgName*, is a collection of 116 organization names. (Thanks to Nick Kushmeric for providing this data). We consider two clusterings of this data, one into 56 clusters, and one into 60 clusters: the difference is that in the second clustering, different branches of an organization (for example: *Virginia Polytechic Institute, Blacksburg* and *Virginia Polytechnic Institute, Charlottesville*) are considered distinct, and in the first, they are not. We used two constrained matching datasets. The *Restaurant* dataset contains 533 restaurants from one restaurant guide to be matched with 331 from a second guide. (Thanks to Sheila Tejada for providing this data.) The *Parks* dataset contains 388 national park names from one listing and 258 from a second listing, with 241 names in common.

Clustering Domains	Split Sizes	#Clusters	Canopy settings T^{tight}/ T^{loose}	Potential Recall	#Positive Pairs	#Negative Pairs
Cora	991	65	0.36 0.53	0.972	19111	7379
	925	64		0.998	15431	8711
OrgName 1	60	42	0.24 0.40	1.000	33	56
	56	17		1.000	196	250
OrgName 2	53	34	0.24 0.40	1.000	36	48
	63	22		1.000	270	181
Matching Domains	Split Sizes	#Pairs	Canopy settings T^{tight} T^{loose}	Potential Recall	# Positive Pairs	#Negative Pairs
Restaurant	430	52	0.28 0.93	1.000	52	426
	434	60		0.983	59	153
Park names	325	124	0.30 0.90	0.992	124	304
	321	117		0.975	117	357

Table 1. Experimental domains.

For most datasets we constrain 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 by exploring a variety of cluster sizes. Here we tried two values of *k* and report the one which gave the best F1 value: this was obtained by constraining the system to create 1.5 times the true number of clusters

To evaluate the algorithms we used a variation of two-fold cross-validation. 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. We will report precision, recall, and F1 results for each partition. To keep the two partitions separate, we split the data so that no algorithm that considers only pairs in the PossiblePairs set produced by the canopy algorithm would ever consider a pair containing one instance from the test set and one instance from the training set. This also implies that no training set instance is within T^{loose} of any test set instance. A disadvantage of this procedure is it is sometimes impossible to create well-balanced splits, biasing the results away from adaptive methods.

The results of our experiments are shown in Table 2, below. We record for each dataset the number of entity names in each partition; the number of desired clusters or pairs; the thresholds used for the canopy algorithm; the maximum recall obtainable using the PossiblePairs produced by the canopy algorithm; and number of positive and negative example pairs given to the learner.

	TFIDF			Edit distance			Adaptive		
	Precis.	Recall	F1	Precis.	Recall	F1	Precis.	Recall	F1
Cora	0.675	0.845	0.751	0.742	0.965	0.839	0.985	0.907	**0.945**
	0.605	0.893	0.721				0.994	0.936	**0.964**
OrgName1		0.939	**0.925**	0.538	0.424	0.633	0.938	0.909	0.923

	0.912 0.237	0.799	0.366	0.935	0.966	**0.950**	0.712	0.852	0.776
OrgName 2	0.971 0.659	0.944 0.951	**0.958** 0.778	0.667 0.862	0.5 0.968	0.571 0.912	0.971 0.996	0.944 0.971	**0.958** **0.984**
Restaurant	0.981 0.967	0.981 0.967	0.981 **0.967**	0.827 0.867	0.827 0.867	0.827 0.867	1.000 0.95	1.000 0.95	**1.000** 0.950
Parks	0.976 0.967	0.976 0.967	0.976 **0.967**	0.967 0.967	0.967 0.967	0.967 0.967	0.984 0.967	0.984 0.967	**0.984** **0.967**

Table 2. Experimental results. Results for Cora/edit distance are from [MNU2000].

The baseline results for edit distance are taken from [MNU2000], who used hand-tuned edit distance, and also report results over the whole set. For each problem, the best obtained F1 measure is placed in bold. On nine of the ten partitions, the adaptive method obtains results comparable to or better than the best of the baseline approaches. We conjecture that the relatively poor performance of the adaptive system on OrgName1 is due to variation across the two partitions used for training and testing.

6 Related Work

Our formalization of adaptive clustering and matching is inspired by the model of "learning to order" of Cohen, Schapire, and Singer [CSS1999]. They consider adaptive systems where, in an example *(x,y), x* is a set of objects to be ordered, and *y* is an intended ordering, 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 give provable bounds on the loss of such 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 Nigam, McCallum and Unger [NMU1999]. However, in our system, we consider matching as well as clustering, and more importantly, 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. [HS95,ME96]) or general frameworks for manually implementing similarity metrics (e.g.,[GFS2000]).

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., [KA1985]) much of which based on a record-linkage theory proposed by Felligi and Sunter [FS1969]. 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. 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 matching, and application of the pairing function to both clustering and constrained matching.

7 Concluding remarks

We have presented a new scalable adaptive scheme for clustering or matching. Experimental results with the method are comparable to or better than clustering or matching with two plausible

fixed distance metrics. 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. Other features might also be useful; e.g., following [BS1998], performance could likely be improved by adding features based on word types. We are in the process of adding features for locations, names, and common words. We also hope explore other approaches to adaptation. We also hope to investigate the robustness of the method with respect to the size of the training data and the number of clusters, and explore semi-supervised learning approaches.

References

[[DLR1977] A. Dempster, N Laird, and D. Rubin (1977). *Using probabilistic models of document retrieval without relevance information*, Journal of the Royal Statistical Society, **39**(B), pages. 1-38.

[A1973] Anderberg, M.. R. (1973). *Cluster Analysis for Application*. Academic Press.

[BH1994]. J. Buhmann and T. Hofmann. (1994) *Central and pairwise data clustering by competitive neural networks*. In J. Cowan, G. Tesauro, and J. Alspector, editors, Advances in Neural Information Processing Systems 6, pages 104--111. Morgan Kaufmann, San Francisco, CA.

[BS1998] Borthwick, A., Sterling, J., Agichtein, E., and Grishman R. (1998). NYU: Description of the MENE Named Entity System as used in MUC-7. *Proc. of the Seventh Message Understanding Conference (MUC-7)*, Fairfax, VA, April 29 - May 1, 1998.

[C2000] Cohen, W.W. (2000). *Data Integration using Similarity Joins and a Word-based Information Representation Language*, ACM Transactions on Information Systems, 18(3), July 2000, pp 288-321 .

[CSS1999] Cohen, W.W., Schapire, R.E., and Singer, Y. (1999). *Learning to Order Things*, Journal of AI Research, Volume 10, pages 243-270.

[DLR1977] A. Dempster, N Laird, and D. Rubin (1977). *Using probabilistic models of document retrieval without relevance information*, Journal of the Royal Statistical Society, **39**(B), pages. 1-38.

[FS1969] I. P. Felligi and A. B. Sunter. *A theory for record linkage*. Journal of the American Statistical Society, 64:1183-1210, 1969.

[GC1998] J. Goldstein and J. Carbonell. *The use of MMR and diversity-based reranking in document reranking and summarization*. In Proceedings of the 14th Workshop on Language Technology in Multimedia Information Retrieval, pages 152--166, Enschede, the Netherlands.

[GFS2000] Galhardas H., Florescu D., Shasha D., and E. Simon. (2000). *Ajax: An Extensible Data Cleaning Tool*. In Proceedings of ACM SIGMOD-2000, June 2000.

[GWW199] Y. Gdalyahu, D. Weinshall, and M. Werma. (1999) *A randomized algorithm for pairwise clustering,* in *Advances in Neural Information Processing Systems* (M. Kearns, S. Solla, and D. Cohn, eds.), vol. 11, MIT Press, USA, 1999.

[HS1998] Hernandez, M. A. and Stolfo, J. S.: "*Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem*", Journal of Data Mining and Knowledge Discovery, No. 2, 1998, pp. 9-37

[KA1985] Kilss B. and Alvey W., editors. Record Linkage Techniques|1985, 1985. *Statistics of Income Division*, Internal Revenue Service Publication 1299-2-96. Available from http://www.fcsm.gov/

[LGB1999] Lawrence, S. Giles, C.L., Bollacker, K. D. (1999). *Autonomous Citation Matching*, Proceedings of the Third International Conference on Autonomous Agents, Seattle, Washington, May 1-5, ACM Press.

[ME1996] Monge A. and Elkan C.. *The field-matching problem: algorithm and applications*. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, August 1996

[MNRS2000] McCallum, A., Nigam, K., Rennie, J., & Seymore, K. (2000). *Automating the Construction of Internet Portals with Machine Learning*. Information Retrieval.

[MNU2000] McCallum, A., Nigam, K., Ungar, L. (2000). *Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching*. Knowledge Discovery and Data Mining, pages 169-178.

[S1989] Salton, G. (1989). Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley.

[SL1993] H.A. Baier Saip, C.L. Lucchesi (1993). *Matching algorithms for bipartite graph*, Technical Report DCC-03/93 (Departamento de Ciência da Computação, Universidade Estudal de Campinas).

[SSMB1997] Salton, G., Singhal, A., Mitra, M. & Buckley, C. (1997). *Automatic text structuring and summarization*. In Information Processing and Management, Elsevier Science, 33, 193-207.

[WF1974] R.A. Wagner, M.J. Fischer (1974). *The string-to-string correction problem*, J. ACM 21, pages 168-178.

[YC1994] Y. Yang and C. G. Chute. (1994) *An example-based mapping method for text classification and retrieval,* ACM Transactions on Information Systems,12(3).