PhD Proposal, Oct 2003 (Also appears as Technical Report, UT-AI-TR-03-305)

Learnable Similarity Functions and Their Applications to Record Linkage and Clustering

Mikhail Bilenko
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
mbilenko@cs.utexas.edu

Doctoral Dissertation Proposal

Supervising Professor: Raymond J. Mooney

Abstract

Many machine learning tasks require similarity functions that estimate likeness between observations. Similarity computations are particularly important for clustering and record linkage algorithms that depend on accurate estimates of the distance between datapoints. However, standard measures such as string edit distance and Euclidean distance often fail to capture an appropriate notion of similarity for a particular domain or dataset. This problem can be alleviated by employing *learnable* similarity functions that adapt using training data. In this proposal, we introduce two adaptive string similarity measures: (1) Learnable Edit Distance with Affine Gaps, and (2) Learnable Vector-Space Similarity Based on Pairwise Classification. These similarity functions can be trained using a corpus of labeled pairs of equivalent and non-equivalent strings. We illustrate the accuracy improvements obtained with these measures using MARLIN, our system for record linkage in databases that learns to combine adaptive and static string similarity functions in a two-level learning framework.

Obtaining useful training examples for learnable similarity functions can be problematic due to scarcity of informative similar and dissimilar object pairs. We propose two strategies, Static-Active Selection and Weakly-Labeled Selection, that facilitate efficient training data collection for record linkage. Additionally, we describe a method for combining seeding with Euclidean distance learning for semi-supervised k-means clustering. Experimental evaluation demonstrates that our method outperforms unsupervised clustering and semi-supervised clustering that employs seeding or metric learning separately.

In future research, we intend to pursue several directions in developing accurate learnable similarity functions and applying them to record linkage and clustering problems. This work will involve improving the proposed string similarity functions as well as introducing several novel approaches to adaptive string distance computation. We also plan to extend our initial work on learnable similarity functions for clustering, particularly for high-dimensional data. Finally, we will investigate the utility of various active learning strategies for learning similarity functions, as well as extend the preliminary work on static-active selection of training pairs.

Contents

1 Introduction

Many problems in machine learning and data mining depend on distance estimates between observations. Consequently, a large number of functions that estimate similarity between objects have been developed for different data types, varying greatly in their expressiveness, mathematical properties, and assumptions.

However, the notion of similarity can differ depending on the particular domain, dataset, or task at hand. Similarity between certain features may be highly indicative of overall object similarity, while other features may be unimportant. A number of similarity functions have been proposed that combine similarity between individual features in various ways (Tversky, 1977; Gusfield, 1997; Baxter, 1997; Baeza-Yates & Ribeiro-Neto, 1999). Additionally, there exists a substantial body of research on feature space transformations that attempt to provide a more salient representation of data than the original feature space, e.g. Principal Component Analysis (PCA) and Independent Component Analysis (ICA) methods (Jolliffe, 1986; Hyvärinen & Oja, 2000).

All of these techniques make certain assumptions about the optimal representation of data and its influence on computing similarity which may or may not be applicable for specific datasets and tasks. Therefore, it is desirable to *learn* similarity functions from training data to capture the correct notion of distance for a particular task at hand in a given domain. Recently, several approaches have been suggested for training such functions that use *pairwise constraints* in the form of pairs of datapoints labeled as similar or dissimilar (alternatively, pairs can be composed of points that are equivalent or distinct) (Ristad & Yianilos, 1998; Phillips, 1999; Cohen & Richman, 2002; Xing, Ng, Jordan, & Russell, 2003). These approaches have shown improvements over traditional similarity functions for different data types such as vectors in Euclidean space, strings, and database records composed of multiple text fields. While these initial results are encouraging, there still remains a large number of similarity functions that are currently unable to adapt to a particular domain. In our work, we attempt to bridge this gap by developing both new learnable similarity functions and methods for their application to particular problems in machine learning and data mining.

We begin by introducing two learnable similarity functions for strings. The first one is based on a probabilistic model of edit distance with affine gaps (Gotoh, 1982), a widely used character-based metric. The second one is a vector-space similarity function that utilizes a Support Vector Machine (SVM) classifier (Vapnik, 1998) to discriminate between similar and dissimilar string pairs. As opposed to their *static* analogs, these similarity functions *adapt* to a particular domain using training examples and produce more accurate similarity estimates as a result.

We apply these functions to the *record linkage* problem, which is the task of identifying semantically equivalent database records that are syntactically different (Newcombe, Kennedy, Axford, & James, 1959). Record linkage is one of the key problems for data cleaning since presence of unidentified duplicate records violates data integrity principles. Information integration from multiple sources also relies on record linkage methods because information describing the same entity must be linked across sources. Record linkage algorithms fundamentally depend on string similarity functions for discriminating between equivalent and non-equivalent record fields, as well as on record similarity functions for combining similarity estimates from individual string fields. We use record linkage as a testbed for evaluating learnable string metrics: our system, MARLIN (Multiply Adaptive Record Linkage using INduction), learns to combine trainable string metrics in a two-layer framework for identifying duplicate database records (Bilenko & Mooney, 2003a). In initial evaluation on several real-world datasets we show that learnable similarity functions both at the string level and at the record level lead to higher record linkage accuracy than static approaches.

For any supervised learning task, the learning process can be facilitated significantly by intelligent selection of informative training examples from a pool of unlabeled data, and a number of such active learning methods have been proposed (Lewis & Catlett, 1994; Tong, 2001; Muslea, 2002). Because training data for learnable similarity functions is composed of object pairs, selecting meaningful training examples can be problematic since very few randomly selected object pairs are informative for learning similarity functions, while traditional active learning methods tend to be computationally expensive. Based on our initial experiments with MARLIN, we propose two strategies for selectively collecting training data: static-active sampling and weakly-labeled selection. Experimental evaluation shows that using static-active selection leads to improvements over random selection of training data for similarity function learning without the computational cost of traditional active learning methods. Weakly-labeled selection, on other hand, allows unsupervised collection of negative training examples, reducing the burden of labeling data for the user (Bilenko & Mooney, 2003b).

Another important application that can benefit from using learnable similarity functions is clustering. While traditionally clustering is an unsupervised learning problem, recently there has been increasing attention to semi-supervised clustering, where some supervision is provided to obtain better grouping of data (Cohn, Caruana, & McCallum, 2000; Wagstaff, Cardie, Rogers, & Schroedl, 2001; Basu, Banerjee, & Mooney, 2002; Xing et al., 2003). We present MPC-KMEANS, a method that combines cluster initialization via seeding with learning Euclidean distance for semi-supervised k-means clustering. Our results show that using both similarity metric learning and seeding leads to improvements over both unsupervised clustering and the individual methods in separation (Basu, Bilenko, & Mooney, 2003b).

In future work, we will pursue several directions in developing better adaptive similarity metrics and applying them to duplicate detection and clustering:

- Extending vector-space string similarity based on pairwise classification by incorporating advanced techniques for transforming strings into vector space (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002) and methods for tuning Support Vector Machines for optimal accuracy (Valentini & Dietterich, 2003);
- Incorporating *macro detection and learning* into learnable string edit distance with affine gaps;
- Designing a model for edit distance based on *conditional random fields* (Lafferty, McCallum, & Pereira, 2001), an undirected graphical model that has been shown to be more effective than traditional probabilistic frameworks such as hidden Markov models;
- Developing string distance functions that learn *semantic* string similarity using generic external sources such as large text corpora and the World Wide Web;
- Constructing and evaluating learnable similarity functions for high-dimensional vector-space data that would lead to improved semi-supervised clustering algorithms;
- Applying active learning methods to similarity function training to minimize the amount of labeled data required for adaptation.

We hope that pursuing these research topics will lead to developing a number of learnable similarity functions that can be used in such applications as duplicate detection and clustering.

The remainder of this proposal is organized as follows. Section 2 introduces several popular similarity functions for strings and Euclidean space vectors, as well as prior work on unsupervised and supervised learning of similarity functions. Duplicate detection and clustering problems are also described in this section. Section 3 presents two new learnable string similarity functions, and describes MARLIN, our duplicate detection system, as well as MPC-KMEANS, our method for semi-supervised k-means clustering. We discuss proposed directions for future work in Section 4.

2 Background and Related Work

Because many supervised and unsupervised learning algorithms require estimating similarity between objects, a number of distance functions for various object description types have been developed. In this section, we provide a brief overview of most important distance functions for text and vector-space data, and describe prior work on algorithms for adapting these functions to particular domains. We also provide background on two important problems, record linkage and clustering, most algorithms for which rely on similarity estimates between observations. We believe that these two problems can benefit greatly from employing learnable distance functions, and we will use them for evaluating our adaptive methods. Finally, we introduce active learning methods that select informative training examples from a pool of unlabeled data.

Let us briefly describe the notation that we will use in the rest of this proposal. Strings are denoted by lower-case italic letters such as s and t; brackets are used for string characters and subsequences: $s[i]$ stands for i-th character of string s, while $s[i{:}j]$ represents the contiguous subsequence of s from i-th to j-th character. We use lowercase bold letters such as $\mathbf{x}$ and $\mathbf{y}$ for vectors, and uppercase bold letters for matrices such as $\mathbf{\Sigma}$ and $\mathbf{U}$. Sets are denoted by script uppercase letters such as $\mathcal{S}$ and $\mathcal{V}$.

2.1 String Similarity Functions

Techniques for calculating similarity between strings can be roughly separated into two broad groups: sequence-based functions and token-based functions. Sequence-based functions model string similarity by viewing strings as contiguous sequences that differ at the level of individual characters. Token-based functions, on other hand, do not view strings as contiguous sequences but as unordered sets of tokens. Below we describe most commonly used string similarity measures from both of these families as well as several hybrid distance functions; see (Gusfield, 1997) and (Cohen, Ravikumar, & Fienberg, 2003) for more discussion of string distance measures.

2.1.1 Sequence-based String Similarity Functions

String Edit Distance. The most well-known sequence-based string similarity measure is string edit distance, also known in its simplest form as Levenshtein distance. It defines distance between two strings as the minimum number of character insertions, deletions or substitutions required to transform one string into another (Levenshtein, 1966). For example, consider calculating distance between strings *s = "12 8 Street"* and *t = "12 8th Str."*. The minimum sequence of edit operations that transforms s into t consists of the following five operations, implying that Levenshtein distance between s and t is 5:

1. Insert *"t"*: *"12 8 Street"→"12 8t Street"*
2. Insert *"h"*: *"12 8t Street"→"12 8th Street"*
3. Substitute *"e"* with ".": *"12 8th Street"→"12 8th Str.et'*
4. Delete *"e"*: *"12 8th Str.et"→"12 8th Str.t"*
5. Delete *"t"*: *"12 8th Str.t"→"12 8th Str."*

Needleman and Wunsch (1970) described a general dynamic programming method for computing edit distance using individual edit operations costs. The algorithm computes the distance in $O(|s||t|)$ computational time by recursively calculating distance $D(s[1:i], t[1:j])$ between prefixes of length i and j:

$$D(s[1{:}i], t[1{:}j]) = \min \begin{cases} D(s[1{:}i-1], t[1{:}j-1]) + c(s[i], t[j]) \\ D(s[1{:}i-1], t[1{:}j]) + c(s[i], \epsilon) \\ D(s[1{:}i], t[1{:}j-1]) + c(\epsilon, t[j]) \end{cases}$$

where $c(s[i], t[j])$ is the cost of substituting i-th character of s with j-th character of t (which is usually set to 0 or a negative value if the characters are equivalent); while $c(s[i], \epsilon)$ and $c(\epsilon, t[j])$ are the costs of deleting i-th character of s and inserting j-th character of t respectively. The recursive computation of $D(s[1{:}i], t[1{:}j])$ corresponds to a matrix of prefix matching costs as well as to an *alignment* of characters of s and t. Figure 1 below illustrates the D matrix and the optimal alignment for the above example assuming unit costs for all insertions, deletions, and substitutions, and zero cost for exact matching.

		1	***2***	_	***8***	***t***	***h***	_	***S***	***t***	***r***	**.**
	0	1	2	3	4	5	6	7	8	9	10	11
1	1	**0**	1	2	3	4	5	6	7	8	9	10
2	2	1	**0**	1	2	3	4	5	6	7	8	9
_	3	2	1	**0**	1	2	3	4	5	6	7	8
8	4	3	2	1	**0**	**1**	**2**	3	4	5	6	7
_	5	4	3	2	1	1	2	**2**	3	4	5	6
S	6	5	4	3	2	2	2	3	**2**	3	4	5
t	7	6	5	4	3	2	3	3	3	**2**	3	4
r	8	7	6	5	4	3	3	4	4	3	**2**	3
e	9	8	7	6	5	4	4	4	5	4	3	**3**
e	10	9	8	7	6	5	5	5	5	5	4	**4**
t	11	10	9	8	7	6	6	6	6	5	5	$\boxed{\mathbf{5}}$

(a)

1	2	_	8	▼	▼	_	S	t	r	e	e	t
1	2	_	8	t	h	_	S	t	r	.	▲	▲

(b)

Figure 1: (a) Dynamic programming computation of Levenshtein distance between strings *"12 8 Street"* and *"12 8th Str."*. (b) Optimal alignment of the strings, corresponding to the sequence of edit operations that can be traced by following the matrix elements shown in bold font above.

Two extensions to string edit distance have been suggested that lead to a significantly more flexible distance function. First, Smith and Waterman (1981) proposed modifying the above computation to discard mismatching regions of two strings and only compute similarity of matching regions. Second, Gotoh (1982) suggested penalizing gaps, or contiguous inserted or deleted regions, using a linear model: $cost(s[i{:}j]) = \sigma c(s[i], \epsilon) + \delta \sum_{k=i+1\dots j} c(s[k], \epsilon)$, where $s[i{:}j]$ is the contiguous substring of s that is aligned to a gap, σ is the cost of starting a gap and δ is the per-character cost of extending a gap ($\delta < \sigma$). Since similar strings often differ due to abbreviations or whole-word insertions and deletions, this model typically produces more robust similarity estimates than Levenshtein distance. The resulting metric, string distance with affine gaps $D_{ag}(s, t)$, can be computed using a dynamic programming algorithm that constructs three matrices based on the following recurrences in $O(|s||t|)$ computational time:

$$M(s[1{:}i], t[1{:}j]) = \min \begin{cases} M(s[1{:}i-1], t[1{:}j-1]) + c(s[i], t[j]) \\ I(s[1{:}i-1], t[1{:}j-1]) + c(s[i], t[j]) \\ D(s[1{:}i-1], t[1{:}j-1]) + c(s[i], t[j]) \end{cases}$$

$$D(s[1{:}i], t[1{:}j]) = \min \begin{cases} M(s[1{:}i-1], t[1{:}j]) + \sigma + c(s[i], \epsilon) \\ D(s[1{:}i-1], t[1{:}j]) + \delta + c(s[i], \epsilon) \end{cases} \tag{1}$$

$$I(s[1{:}i], t[1{:}j]) = \min \begin{cases} M(s[1{:}i], t[1{:}j-1]) + \sigma + c(\epsilon, t[j]) \\ I(s[1{:}i], t[1{:}j-1]) + \sigma + c(\epsilon, t[j]) \end{cases}$$

$$D_{ag}(s,t) = \min(I(s,t), D(s,t), M(s,t))$$

In this recurrent computation, each matrix element $M(s[i], t[j])$ contains the distance between prefixes $s[1{:}i]$ and $t[1{:}j]$ for the minimum-cost alignment that ends with the last two characters of the prefixes, $s[i]$ and $t[j]$, being matched. Matrix elements $I(s[i], t[j])$ and $D(s[i], t[j])$ give the distances between prefix alignments that end in insertion and deletion gaps respectively. Costs of edit operations $c(s[i], t[j])$ are defined as for Levenshtein distance above. The final distance between the strings is the minimum of three alignments: $M(s,t)$ matching the last characters of the two strings, $D(s,t)$ matching the last character of the first string with a gap in the second string, and $I(s,t)$ matching the last character of the second string with a gap in the first string.

String edit distance with affine gaps is widely used in practice in a variety of applications ranging from biological sequence analysis (Durbin, Eddy, Krogh, & Mitchison, 1998) to duplicate detection (Monge & Elkan, 1997). It is particularly useful for comparing short and medium-length strings in domains where corruptions at the level of individual characters are common.

Jaro-Winkler String Similarity. Researchers working on the problem of identifying matching names in census records have created several efficient string similarity functions specialized for the matching task. Jaro (1989) proposed a metric that is less sensitive to character transpositions. Given strings s and t, character $s[i]$ is defined to be *common* to s and t if there exists $t[j] = s[i]$ such that $i - H \leq j \leq i + H$, where $H = \min(|s|, |t|)$. Given sequences of common characters s' of s and t' of t, the *number of transpositions* $T(s', t')$ between s and t is defined as one-half of the number of characters in s' and t' that do not match ($s'[i] \neq t'[i]$). Then, the Jaro metric can be computed as:

$$Sim_{Jaro}(s,t) = w_1 \frac{|s'|}{|s|} + w_2 \frac{|t'|}{|t|} + w_\tau \frac{|s'| - T(s', t')}{|s'|}$$

where weights w_1, w_2, and w_τ are typically set to $\frac{1}{3}$. Several enhancements to the Jaro metric have been shown to increase its accuracy (Winkler, 1994), most important ones of which are:

- Counting disagreeing but similar characters (e.g. *"1"* and *"l"*) as contributing weight 0.3 to the number of common characters $C(s,t)$;
- Increasing similarity value proportionally to exact agreement in string prefixes (Winkler, 1990).

The resulting similarity function, known as Jaro-Winkler similarity, is computed as:

$$Sim_{JW}(s,t) = Sim_{Jaro}(s,t) + \frac{\max(P(s,t), 4)}{10}(1 - Sim_{Jaro}(s,t)) \tag{2}$$

where $P(s,t)$ is the length of the longest common prefix of s and t.

The Jaro-Winkler metric is widely used for matching census records (Winkler, 1994), and has also been shown to be competitive with edit distance for the task of matching equivalent database records composed of short strings (Cohen et al., 2003).

2.1.2 Token-based String Similarity Functions

While sequence-based string similarity functions work well for estimating distance between shorter strings that differ largely at character level, they become too computationally expensive and less accurate for larger strings. For example, when differences between equivalent strings are due to insertions, deletions and transpositions of multiple words, sequence-based similarity functions assign high cost to non-aligned string segments, resulting in low similarity scores for strings that share many common words. At the same time, computing string edit distance becomes computationally prohibitive for larger strings such as text documents on the Web since the computational complexity is quadratic in average string size.

The vector-space model of text avoids these problems by viewing strings as "bags of tokens" and disregarding the order in which the tokens occur in the strings (Salton & McGill, 1983). Jaccard similarity can then be used as the simplest method for computing likeness as the proportion of tokens shared by both strings. If strings s and t are represented by sets of tokens $\mathcal{S}$ and $\mathcal{T}$, Jaccard similarity is:

$$Sim_{Jaccard} = \frac{|\mathcal{S} \cap \mathcal{T}|}{|\mathcal{S} \cup \mathcal{T}|} \tag{3}$$

The primary problem with Jaccard similarity is that it does not take into account the relative importance of different tokens. Tokens that occur frequently in a given string should have higher contribution to similarity than those that occur few times, as should those tokens that are rare among the set of strings under consideration. The Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme achieves this by associating a weight $w_{v_i,s} = \frac{N(v_i,s)}{\max_{v_j \in s} N(v_j,s)} \cdot \log \frac{N}{N(v_i)}$ with every token v_i from string s, where $N(v_i, s)$ is the number of times v_i occurred in s (term frequency), N is the number of strings in the overall corpus under consideration, and $N(v_i)$ is the number of strings in the corpus that include v_i (document frequency).

Given a corpus of strings that yields the set $\mathcal{V}$ of distinct tokens after tokenization, a string s can be represented as a $|\mathcal{V}|$-dimensional vector of weights $w_{v_i,s}$, every non-zero component of which corresponds to a token present in s. TF-IDF cosine similarity between two strings is defined as the cosine of the angle between their vector representations:

$$Sim_{TF-IDF}(s,t) = \frac{\sum_{v_i \in \mathcal{V}} w_{v_i,s} w_{v_i,t}}{\sqrt{\sum_{s_i \in \mathcal{S}} w_{s_i,s}^2} \cdot \sqrt{\sum_{t_i \in \mathcal{T}} w_{t_i,t}^2}} \tag{4}$$

With the help of appropriate hashing data structures, TF-IDF cosine similarity is computationally efficient due to high sparsity of most vectors, and provides a reasonable off-the-shelf metric for long strings and text documents. Tokenization is typically performed by treating each individual word of certain minimum length as a separate token, usually excluding a fixed set of functional "stopwords" and optionally stemming tokens to their roots (Baeza-Yates & Ribeiro-Neto, 1999). An alternative tokenization scheme is known as n-grams: it relies on using all overlapping contiguous subsequences of length n as tokens.

2.1.3 Hybrid String Similarity Functions

Recently, several hybrid string similarity functions were introduced that combine sequence-based and token-based methods. Monge and Elkan (1996) proposed a distance function that applies some "secondary" sequence-based similarity function $Sim'(s,t)$ for each combination of tokens from the two strings and takes the average of maximum values for each token . Given two strings s and t and their token sets $\mathcal{S}$ and $\mathcal{T}$, similarity between s and t is defined as:

$$Sim_{ME}(s,t) = \frac{1}{|\mathcal{S}|} \sum_{s_i \in \mathcal{S}} \max_{t_j \in \mathcal{T}} Sim'(s_i, t_j) \tag{5}$$

Another hybrid string similarity function, "soft" TF-IDF similarity, was proposed by Cohen et al. (2003). They also employ a "secondary" sequence-based similarity function $Sim'(s,t)$ to identify tokens in the two strings that are sufficiently similar according to Sim', and then incorporate similarity between these tokens into the computation of TF-IDF vector-space similarity:

$$Sim_{SoftTFIDF}(s,t) = \sum_{v \in CLOSE(\theta,s,t)} w_{v,s} \cdot N(v,t) \cdot Sim'(w,t) \tag{6}$$

where $CLOSE(\theta, s, t)$ is the set of tokens v from string s for which there exists some token v' in string t such that $Sim'(v, v') > \theta$, and $N(v,t) = \max_{v' \in t} Sim'(v, v')$. Cohen et al. (2003) have shown that this hybrid metric that uses the Jaro-Winkler metric Sim_{JW} as the "secondary" similarity function frequently outperforms traditional sequence-based and token-based metrics on the task of matching equivalent strings, thus combining the strengths of these two families of similarity functions.

2.2 Vector-space Similarity Functions

In Euclidean space, the Minkowski family of metrics, also known as the L_k norms, includes most commonly used similarity measures for objects described by d-dimensional vectors:

$$L_k(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^k \right)^{1/k} \tag{7}$$

The L_2 norm, commonly known as Euclidean distance, is frequently used for low-dimensional vector data. Its popularity is due to a number of factors:

- Intuitive simplicity: the L_2 norm corresponds to straight-line distance between points in Euclidean space;
- Invariance to rotation or translation in feature space;
- Mathematical metric properties: non-negativity ($L_2(\mathbf{x}, \mathbf{y}) \geq 0)$), reflexivity ($L_2(\mathbf{x}, \mathbf{y}) = 0$ iff $\mathbf{x} = \mathbf{y}$), symmetry ($L_2(\mathbf{x}, \mathbf{y}) = L_2(\mathbf{y}, \mathbf{x})$), and triangle inequality ($L_2(\mathbf{x}, \mathbf{y}) + L_2(\mathbf{y}, \mathbf{z}) \geq L_2(\mathbf{x}, \mathbf{z})$).

Despite these attractive characteristics, Euclidean distance as well as other Minkowski metrics suffer from the *curse of dimensionality* when they are applied to high-dimensional data (Friedman, 1997). As the dimensionality of the Euclidean space increases, sparsity of observations increases exponentially with the number of dimensions, which leads to observations becoming equidistant in terms of Euclidean distance. Cosine similarity, or normalized dot product, has been widely used as an alternative similarity function for high-dimensional data (Duda, Hart, & Stork, 2001):

$$Sim_{cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T \mathbf{y}}{\|x\| \|y\|} = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}} \tag{8}$$

If applied to normalized vectors, cosine similarity obeys metric properties; in general, however, it is not a metric in the mathematical sense, and it is not invariant to translations and linear transformations.

2.3 Methods for Unsupervised Learning of Similarity Functions

Given a set of observations, it may be possible to obtain insights about relative importance of different features by observing statistical properties of the dataset. There exists a number of techniques that modify either the distance function computation or the data representation to provide more meaningful measures of similarity. The TF-IDF weighting scheme described above can be viewed as one such method for string similarity since it takes statistical properties of the entire dataset into account when performing computations. Below we describe some of the many such methods for data in Euclidean space.

2.3.1 Mahalanobis distance

Given a set of observation vectors $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, it is possible to scale the computation of Euclidean distance according to the variance of observation components in each dimension. Squared Mahalanobis distance is defined as:

$$D^2_{Mah}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)\mathbf{\Sigma}^{-1}(\mathbf{x}_i - \mathbf{x}_j) \tag{9}$$

where $\mathbf{\Sigma}^{-1}$ is the inverse of the covariance matrix $\mathbf{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$, and $\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$ is the data mean.

Essentially, Mahalanobis distance attempts to give each dimension equal weight when computing distance by scaling its contribution proportionally to variance, while taking into account co-variances between the dimensions.

2.3.2 Space transformation methods

Although Mahalanobis distance "evens out" the contributions of different dimensions to similarity comparisons, it is possible that certain features are redundant and non-informative, while others do not capture the variance in the dataset. In such cases projecting the data onto some lower-dimensional subspace may lead to a better representation in which distance computations using traditional similarity functions can be performed.

Principal component analysis (PCA). Principal component analysis, also known as the Karhunen-Loéve transform, finds a lower-dimensional subspace that can embed the projection of data that is optimal in the linear least-squares sense (Jolliffe, 1986). PCA is performed by computing the covariance matrix $\mathbf{\Sigma}$ for a dataset, and obtaining its eigenvalues and eigenvectors. The eigenvectors with largest eigenvalues correspond to orthogonal directions of highest variance in data. By forming a matrix $\mathbf{A}$ of eigenvectors corresponding to the largest eigenvalues, the original data can be projected into the subspace formed by directions of largest variance: $\mathbf{x}' = \mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu})$.

Independent Component Analysis. Independent component analysis is a recently developed technique for finding the statistically independent components of a vector of observed signals (Comon, 1994). Given a d-dimensional vector $\mathbf{x}$, ICA tries to find a full-rank $d \times r$ matrix A and an r-dimensional vector $\mathbf{s}$ such that $x = As$ where components of s are statistically independent. Intuitively, ICA can be understood through its relation to the blind source separation problem: each independent component can be viewed as a source of data with a separate set of parameters. A number of ICA algorithms exists based on various techniques including fixed-point iteration and maximum likelihood; see (Hyvärinen & Oja, 2000) for an overview.

Non-negative Matrix Factorization (NMF). Non-negative matrix factorization is distinguished from other feature space reduction techniques by its use of non-negativity constraints (Lee & Seung, 1997). These constraints lead to a parts-based representation because they allow only additive combinations, which is compatible with the intuitive notion of combining parts to form a whole. The NMF factorization is usually found using iterative algorithms based on several proposed objective functions, see (Lee & Seung, 1997) for details. NMF has been shown to be successful in learning parts of faces and semantic features of text (Lee & Seung, 1999). However, good approximation can only be achieved if the basis vectors discover structure that is latent in the data. For example, when using NMF to analyze face images, the basis vectors would contain non-global information about different facial parts. The data vectors can be reconstructed by combining these different parts. Because both the basis and encodings are typically sparse, NMF creates a sparsely distributed encoding of the original data, in contrast to the fully distributed ICA and PCA encodings.

2.4 Methods for Supervised Learning of Similarity Functions

While statistical properties of a collection of observations can be used to extract certain features that are statistically salient, representation obtained by the unsupervised methods described above may not be optimal with respect to the actual user task such as record linkage or clustering. For example, irrelevant features may have the highest variance, and methods such as PCA would result in a representation in which these features are most salient, while more important features with lower variance are under-expressed.

If user supervision in the form of observation pairs that are labeled as similar or dissimilar is available, it is desirable to adapt similarity functions using such feedback directly. Below we describe several such methods for supervised similarity function learning.

2.4.1 Learnable String Edit Distance

Ristad and Yianilos (1998) proposed learning the parameters of string edit distance using a probabilistic generative model. In their model, a string alignment is equivalent to a sequence of character pairs generated by edit operations emitted by a hidden Markov model with a single non-terminal state. Each edit operation corresponds to the probability of observing a pair of characters, or a single inserted/deleted character in the alignment. Given two strings, the model can be used to compute similarity between them as either the probability of the most likely alignment between the strings (using the Viterbi algorithm), or the overall probability of generating the string pair over all possible alignments (Rabiner, 1989).

Given a corpus of matched string pairs, the model parameters can be learned using a variant of the Expectation-Maximization algorithm. Ristad and Yianilos (1998) demonstrate the improvements obtained using their method over Levenshtein distance on the task of matching natural language strings and their phonetic representation.

2.4.2 Learnable Vector-space Similarity Functions

The problem of adapting similarity computations to a given domain using training data has been addressed for Euclidean space data mainly in the context of classification, although recently there has been some work on learning similarity measures for clustering. Supervised methods for learning similarity functions have received particular attention in the context of k-nearest neighbor (k-NN) classifiers. Given an unlabeled observation vector $\mathbf{x}$, these classifiers use the majority vote of k training points that are nearest to $\mathbf{x}$ to categorize $\mathbf{x}$. Because determining which points are nearest directly relies on accurately computing similarity, several methods have been proposed for k-NN classifiers that allow using flexible similarity metrics which attempt to identify most meaningful features and give them higher weight.

Hastie and Tibshirani (1996) proposed an algorithm, Discriminative Adaptive Nearest Neighbor (DANN), that combines k-nearest neighbor with linear discriminant analysis. This technique relies on computing a weighting matrix $\mathbf{\Sigma} = \mathbf{W}^{-1/2}(\mathbf{W}^{-1/2}\mathbf{B}\mathbf{W}^{-1/2} + \epsilon\mathbf{I})\mathbf{W}^{-1/2}$, where $\mathbf{B}$ and $\mathbf{W}$ are between-class and within-class covariance matrices respectively. For each test point, $\mathbf{\Sigma}$ is used to weight the Euclidean distance metric that identifies k nearest neighbors.

Friedman (1994) and Domeniconi, Peng, and Gunopulos (2001) developed methods for learning weights for Euclidean distance locally for different points in the context of k-nearest neighbor using class posterior probabilities for data points to compute similarity. In following work, Domeniconi and Gunopulos (2002) proposed an alternative approach to learning local feature relevance by identifying most discriminant directions using the margin of a support vector machine.

Phillips (1999) also used support vector machines for implicit learning of a similarity function on the task of face recognition. By mapping the multi-class classification problem to a two-class decision problem in the *difference space*, where the two classes correspond to same-class vector differences $\{\mathbf{x}_i - \mathbf{x}_j | \mathbf{x}_i \sim \mathbf{x}_j\}$ and different-class vector differences $\{\mathbf{x}_i - \mathbf{x}_j | \mathbf{x}_i \nsim \mathbf{x}_j\}$, the SVM learns to distinguish between informative and uninformative features when classifying instances in the difference space.

In unsupervised learning applications such as clustering, no category labels are available for individual observations. However, supervision can be obtained in some cases with respect to pairs of observations in the form of labeled same-cluster and different-cluster pairs. Cohn et al. (2000) proposed using gradient descent for weighting dimensional components of Jensen-Shannon divergence in the context of Expectation-Maximization (EM) clustering. They assume that a user provides supervision in the form of must-link and cannot-link constraints, which are pairs of observations that should be placed in the same or different clusters respectively. These pairs are then used to learn weights that result in a distance function which reduces distance between same-cluster pairs and maximizes distance between different-cluster pairs.

Xing et al. (2003) assumed a similar scenario where pairwise supervision is used to improve the similarity function used for clustering. They utilize weighted Euclidean distance: $D_A(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathbf{A} (\mathbf{x} - \mathbf{y})}$, where $\mathbf{A}$ is a positive semi-definite matrix of weights. Pairwise constraints are used to pose similarity function learning as a convex optimization problem that aims to minimize the total distance between must-link pairs while constraining distance between cannot-link pairs to be larger than an arbitrary constant. A combination of gradient descent and iterative projections was used to learn the weight matrix $\mathbf{A}$ which lead to improvements over using the unlearned Euclidean metric with k-means clustering.

2.5 Record Linkage

The goal of record linkage is to identify database field-values and records that are semantically identical but differ syntactically. Algorithms for this task rely heavily on string similarity functions that estimate likeness between record fields and overall records. Since variations in representation can arise from typographical errors, misspellings, abbreviations, as well as integration of multiple data sources, using string similarity functions that capture the source of variation is essential for accurate identification of approximate duplicates.

A large body of work exists on the record linkage problem: after being introduced in the context of matching medical records by (Newcombe et al., 1959), it was studied under a number of names including merge/purge (Hernández & Stolfo, 1995), heterogeneous database integration (Cohen, 1998), hardening soft databases (Cohen, Kautz, & McAllester, 2000), reference matching (McCallum, Nigam, & Ungar, 2000), de-duplication (Sarawagi & Bhamidipaty, 2002), and entity-name clustering and matching (Cohen &

Richman, 2002). Typically, standard string similarity measures such as edit distance or vector-space cosine similarity are used to determine whether two values or records are alike enough to be equivalent. In recent work, Cohen and Richman (2001) and Tejada, Knoblock, and Minton (2001) proposed trainable similarity functions that *combine* multiple standard similarity functions. By treating individual field similarities as record features and training a classifier that distinguishes between equivalent and non-equivalent records, a record-level similarity function is obtained. This approach is similar to Phillips's (1999) approach to face recognition via the "difference space" described above.

2.6 Clustering

Clustering can be roughly defined as the problem of partitioning a dataset into disjoint groups so that observations belonging to the same cluster are similar, while observations belonging to different clusters are dissimilar. Traditionally, clustering has been viewed as a form of unsupervised learning, since no class labels on the data are provided. In *semi-supervised clustering*, supervision from a user is incorporated in the form of class labels or pairwise constraints on objects which can be used to initialize clusters, guide the clustering process, and improve the clustering algorithm parameters (Demiriz, Bennett, & Embrechts, 1999; Cohn et al., 2000; Basu et al., 2002).

Clustering has been widely studied for several decades, and a great variety of algorithms for clustering exists (Jain, Murty, & Flynn, 1999). Most popular algorithms include agglomerative clustering (Kaufman & Rousseeuw, 1990), hard and soft k-means (Dhillon & Modha, 2001; Basu et al., 2002), and graph-based methods that include spectral algorithms and partitioning methods (Karypis & Kumar, 1998; Strehl, 2002). Similarity functions, however, are central to the clustering problem regardless of the particular algorithm used, since all of them explicitly or implicitly utilize similarity computations between observations and cluster prototypes or between individual observations. (Strehl, Ghosh, & Mooney, 2000) have shown that selecting an appropriate similarity measure for clustering can have significant impact on clustering results; therefore, using learnable similarity functions for semi-supervised clustering is a promising research direction.

In this proposal, we focus on the k-Means algorithm which is based on iterative relocation of cluster centroids to locally minimize the total distance between the data points and the centroids. Given a set of data points $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^m$, let $\{\boldsymbol{\mu}_h\}_{h=1}^K$ represent the k cluster centroids, and l_i be the cluster assignment of a point $\mathbf{x}_i$, where $l_i \in \mathcal{L}$ and $\mathcal{L} = \{1, \dots, K\}$. The Euclidean K-Means algorithm creates a K-partitioning[1] $\{\mathcal{X}_l\}_{l=1}^K$ of $\mathcal{X}$ so that the objective function $\sum_{\mathbf{x}_i \in \mathcal{X}} \|\mathbf{x}_i - \boldsymbol{\mu}_{l_i}\|^2$ is locally minimized. Spherical k-means is a variant of the algorithm that is effective for clustering high-dimensional data by employing dot product similarity as the similarity metric between vectors on a hypersphere (Dhillon & Modha, 2001).

It can be shown that the K-Means algorithm is essentially an EM algorithm on a mixture of K Gaussians under assumptions of identity covariance of the Gaussians, uniform priors of the mixture components and expectation under a particular conditional distribution (Basu et al., 2002). If $\mathcal{X}$ denotes the observed data, Θ denotes the current estimate of the parameter values of the mixture of Gaussians model and $\mathcal{L}$ denotes the missing data, then in the E-step the EM algorithm computes the expected value of the complete-data log-likelihood $\log p(\mathcal{X}, \mathcal{L}|\Theta)$ over the conditional distribution $p(\mathcal{L}|\mathcal{X}, \Theta)$ (Bilmes, 1997). Maximizing the complete data log-likelihood under the assumptions specified above can be shown to be equivalent to minimizing the K-Means objective function.

[1] K disjoint subsets of $\mathcal{X}$, whose union is $\mathcal{X}$

2.7 Active Learning

When training examples are selected for a learning task at random, they may be suboptimal in the sense that they do not lead to a maximally attainable improvement in performance. *Active learning* methods attempt to identify those examples that lead to maximal accuracy improvements when added to the training set (Lewis & Catlett, 1994; Cohn, Ghahramani, & Jordan, 1996; Tong, 2001). During each round of active learning, the example that is estimated to improve performance the most when added to the training set is identified and labeled. The system is then re-trained on the training set including the newly added labeled example.

Three broad classes of active learning methods exist: (1) uncertainty sampling techniques (Lewis & Catlett, 1994) attempt to identify examples for which the learning algorithm is least certain in its prediction; (2) query-by-committee methods (Seung, Opper, & Sompolinsky, 1992) utilize a committee of learners and use disagreement between committee members as a measure of training examples' informativeness; (3) estimation of error reduction techniques (Lindenbaum, Markovitch, & Rusakov, 1999; Roy & McCallum, 2001) select examples which, when labeled, lead to greatest reduction in error by minimizing prediction variance.

Active learning was shown to be a successful strategy for improving performance using small amounts of training data on a number of tasks, including classification (Cohn et al., 1996), clustering (Hofmann & Buhmann, 1998; Basu, Banerjee, & Mooney, 2003a), and record linkage (Sarawagi & Bhamidipaty, 2002; Tejada, Knoblock, & Minton, 2002).

3 Learnable Similarity Functions

3.1 Motivation

The purpose of every similarity function is to compare its arguments and return a value that measures the degree to which they are alike. The notion of similarity is domain-specific and situation-specific: objects are similar to others only in a certain context (Tversky, 1977). For example, strings *"D. Ivanov"* and *"E. Ivanova"* are likely to describe similar individuals (family members) if they correspond to two records of people residing in a small Texas town. However, if these strings are taken from a phone book in a large Russian city, it is not very likely that the two individuals they describe are connected in any way.

We argue that traditional similarity measures for different data types can be drastically improved if their parameters are *learned* using training data taken from the particular domain. Therefore, the task before us is to propose parameterized similarity functions for different data types and develop algorithms for learning the parameter values from data. In this proposal, we are particularly interested in similarity functions for strings and vectors in Euclidean space, as well as textual records in multi-field databases. In the rest of this section, we describe some preliminary research on adaptive similarity functions for these data types.

3.2 Learnable String Similarity Functions

3.2.1 Learnable edit distance with affine gaps

Different edit operations have varying significance in different domains. For example, a digit substitution makes a major difference in a street address since it effectively changes the house number, while a single letter substitution is semantically insignificant because it is more likely to be caused by a typo or an abbreviation. Frequency and length of abbreviations in string alignment gaps also varies from domain to domain. Therefore, adapting edit distance with affine gaps to a particular domain requires learning the costs for different edit operations and the gap parameters.

We propose a stochastic model for the edit distance with affine gaps similar to the one proposed by Ristad and Yianilos (1998). The affine-gap alignment is modeled by the hidden Markov model shown in Figure 2 below. The three matrices of the original edit distance with affine gaps described in Section 2.1.1 correspond to accumulated forward probabilities for an alignment that ends in one of the three states M, D, or I. A particular alignment of two strings is generated by this model as a sequence of traversals along the edges. Each traversal is accompanied by an emission of a character pair sampled from a probability distribution of the state that is reached via each traversal.

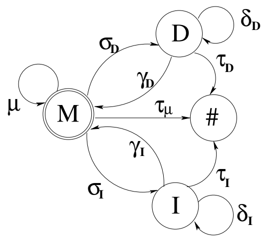

Figure 2: A generative model for edit distance with affine gaps

Given an alphabet of symbols $\mathcal{A}^* = \mathcal{A} \cup \{\epsilon\}$, the full set of edit operations is $\mathcal{E} = \mathcal{E}_s \cup \mathcal{E}_d \cup \mathcal{E}_i$, where $\mathcal{E}_s = \{\langle a, b\rangle \mid a, b \in \mathcal{A}\}$ is the set of all substitution and matching operations $\langle a, b\rangle$; and $\mathcal{E}_i = \{\langle \epsilon, a\rangle \mid a \in A\}$ and $\mathcal{E}_d = \{\langle a, \epsilon\rangle \mid a \in \mathcal{A}\}$ are sets of insertion and deletion operations respectively.

The production starts in state M and terminates when the special state $\#$ is reached. Transitions σ_D and σ_I from the matching state M to either the deletion state D or the insertion state I correspond to a gap in the alignment of the strings. A gap ends when the edge γ_D (or γ_I) is traversed back to the matching state. Remaining in state M by taking edge μ corresponds to a sequence of substitutions or exact matches, while remaining in states I or D is analogous to extending a gap in either the first or the second string. The sum of transition probabilities must be normalized in each state for the model to be complete.

Edit operations emitted in each state correspond to aligned pairs of characters: substitutions $\langle a, b\rangle$ and exact matches $\langle a, a\rangle$ in state M; deletions from the first string $\langle a, \epsilon\rangle$ in state D; and insertions of characters from the second string into the first string $\langle \epsilon, a\rangle$ in state I. Each edit operation $e \in \mathcal{E}$ is assigned a probability $p(e)$ such that $\sum_{e \in \mathcal{E}_s} p(e) = 1$, $\sum_{e \in \mathcal{E}_d} p(e) = 1$, and $\sum_{e \in \mathcal{E}_i} p(e) = 1$. Edit operations with higher probabilities produce character pairs that are likely to be aligned in a given domain, such as substitution $\langle$/, -$\rangle$ for phone numbers, or deletion $\langle$., $\epsilon\rangle$ for addresses.

This generative model is similar to one given for amino-acid sequences in (Durbin et al., 1998) with two differences: (1) transition probabilities are distinct for states D and I, and (2) every transition has a probability parameter associated with it, instead of being expressed through other transitions that are outgoing from the same state.

Given two strings, s and t, probabilities of generating the pair of prefixes $(s[1{:}i], t[1{:}j])$ and suffixes $(s[i{:}|s|], t[j{:}|t|])$ can be computed using dynamic programming in standard forward and backward algorithms in $O(|s||t|)$ time (Rabiner, 1989).

Then, given a corpus of equivalent string pairs $\mathcal{C} = \{(s, t)\}$, this model can be trained using a variant of the Baum-Welch algorithm, shown in Figure 3, which is an Expectation-Maximization procedure for learning parameters of generative models (Rabiner, 1989). The training procedure iterates between two steps, where in the first step the expected number of occurrences for each state transition and edit operation emission is accumulated for a given pair of strings (s, t) from the training corpus. This is achieved by accumulating the posterior probabilities for every possible state transition and an accompanying character pair emission. In the MAXIMIZATION procedure all model parameters are updated using the collected expectations. Complete pseudo-code for the algorithms can be found in (Bilenko & Mooney, 2002).

```
Input: A corpus of equivalent strings C = {(s,t), s ≈ t}
Output: A set of parameters for edit distance with
   affine gaps that minimizes distance for each (s,t) ∈ C
Method:
 until convergence
   for each (s,t) ∈ C
      EXPECTATION-STEP: Use forward and backward algorithms
          to accumulate the expected number of occurrences E[⟨s[i],t[j]⟩]
          for each edit operation used to align s and t,
          as well as E[μ], E[σ_I], E[σ_D], E[δ_I], E[δ_D], E[γ_I], E[γ_D].
   end
   MAXIMIZATION-STEP: Update all transition and emission
      probabilities using the expected numbers of occurrences
      and re-normalize.
 end
```

Figure 3: Training algorithm for generative string distance with affine gaps

It can be proved that this training procedure is guaranteed to converge to a local maximum of likelihood of observing the training corpus $\mathcal{C}$. The trained model can be used for estimating similarity between two strings by computing the probability of generating the aligned pair of strings summed across all possible paths as calculated by the forward and backward algorithms, and then obtaining the posterior probability: $Sim(s,t) = p(s{\sim}t|s,t) = \frac{p(s\sim t,s,t)}{p(s,t)}$. Numerical underflow may occur when these computation are performed for long strings; this problem can be resolved by mapping all computations into logarithmic space or by periodic scaling of all values in matrices M, D and I (Ristad & Yianilos, 1998).

Because the order of strings being aligned usually does not matter for similarity estimation, insertion and deletion operations as well as transitions for states I and D can be represented by a single set of parameters: $p(\langle a, \epsilon\rangle) = p(\langle \epsilon, a\rangle)$ for all symbols $a \in \mathcal{A}$; $\tau{=}\tau_I{=}\tau_D$; $\gamma{=}\gamma_I{=}\gamma_D$; $\delta{=}\delta_I{=}\delta_D$; $\sigma{=}\sigma_I{=}\sigma_D$.

One problem with the described probabilistic model lies in the fact that due to the large size of the edit operation set, probabilities of individual operations are significantly smaller than transition probabilities. If only a few training examples are available, probabilities of some edit operations may be underestimated or even degraded to 0, and distances between strings will vary significantly with minor character variations. To address this problem, the probability distribution over the set of edit operations, $\mathcal{E}$, is smoothed by bounding each edit operation probability by some reasonable minimum value λ.

The learned parameters of the generative distance model can be directly mapped to edit operation costs of the additive model from Section 2.1.1 by taking the negative logarithm of each probability. Distance can then be calculated analogously to Equation (1) with the addition of supplemental costs $g{=}-\log\gamma$ for ending a gap and $m{=}-\log\mu$ for continuing to substitute/match characters. This is equivalent to calculating the cost of the most likely (Viterbi) alignment of the two strings by the generative model in log-space. The resulting similarity function can be viewed as a hybrid between the generative model and the original fixed-cost model.

3.2.2 Learnable Vector-space Similarity

While learnable string edit distance with affine gaps provides us with an adaptive sequence-based string similarity function, there are many domains where token-based similarity functions are preferable. The TF-

IDF weighting scheme is often a good off-the-shelf token-based string similarity function because it attempts to give tokens weights that are proportional to their relative importance. However, the true contribution of each token to similarity is domain-specific. For example, suppose that record linkage is conducted on a database that contains street addresses. If there are several records of addresses from the same street, the street name token (for example, *"34th"*) will have a lower weight due to its many occurrences across the corpus resulting in a lower IDF value. At the same time, some addresses may contain the token *"Square"*, which then will be assigned approximately the same weight as *"34th"* since it may be as common. While two addresses on the same street are more similar than two addresses that are both on squares, the generic TF-IDF cosine similarity is incapable of making this distinction using only statistical information.

If training data in the form of labeled pairs of equivalent and non-equivalent strings is available, it can be used to learn a similarity function that will give more weight to those tokens that are good indicators of similarity. In TF-IDF cosine similarity between two strings shown in Equation (4), every component of the sum corresponds to the i-th token in the vocabulary: $\frac{w_{v_i,s} w_{v_i,t}}{\|s\|\|t\|}$, where $\|s\| = \sqrt{\sum_{s_i \in \mathcal{S}} w_{s_i,s}^2}$ and $\|t\| = \sqrt{\sum_{t_i \in \mathcal{T}} w_{t_i,t}^2}$. Then each component can be thought of as the i-th component of a d-dimensional vector that is being categorized into one of two classes, corresponding to similar and dissimilar strings. Then the vector space similarity computation between two strings can be viewed as two consecutive steps:
(i) A d-dimensional *pair vector* $\mathbf{p}^{(s,t)} = \langle \frac{w_{v_i,s} w_{v_i,t}}{\|s\|\|t\|} \rangle$ is created, each component of which corresponds to the normalized product of weights for the corresponding token in the vocabulary;
(ii) The summation is performed, which is equivalent to the pair instance $\mathbf{p}^{(s,t)}$ being classified by a perceptron with unit weights as either belonging to the equivalent-string class (output is 1), or the different-string class (output is 0). The perceptron output value corresponds to the confidence of the prediction.

Given training data in the form of equivalent-string pairs $\mathcal{S} = \{(s,t),\ s \sim t\}$ and different-string pairs $\mathcal{D} = \{(s,t),\ s \nsim t\}$, a classifier for step (ii) can be trained to produce outputs that correspond to the desired categorization of string pairs. The trained classifier should be able to distinguish between features that are informative for similarity judgments, and those that are not, adapting the computation to a domain-specific notion of similarity contained in the training data.

For this task, a classifier should be able to learn well from limited training sets of high-dimensional data. It also must produce meaningful confidence estimates that correspond to the relative likelihood of belonging to the equivalent-strings or the non-equivalent-strings class. Moreover, the learned hypothesis must be independent of the relative sizes of the positive and negative training sets, since the proportion of equivalent pairs in the training set is likely to be much higher than in the actual database where duplicates are detected.

Support vector machines (Vapnik, 1998) satisfy all of these requirements. SVMs classify input vectors $\mathbf{p}^{(s,t)}$ by implicitly mapping them via a kernel to a high-dimensional space where the two classes ($\mathcal{S}$, equivalent-string pairs, and $\mathcal{D}$, different-string pairs) are separated by a hyperplane. The output of the classifier is the distance between $\phi(\mathbf{p}^{(s,t)})$, the image of the pair vector in high-dimensional space, and the separating hyperplane. This distance provides an intuitive measure of similarity: those pair vectors that are classified with a large margin as belonging to $\mathcal{S}$ correspond to pairs of strings that are highly similar, while those classified with a large margin as belonging to $\mathcal{D}$ represent dissimilar pairs of strings.

The output of an SVM has the form $f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{p_i}, \mathbf{x}) + b$, where $K(\mathbf{x}, \mathbf{y})$ is a kernel function (dot product between images of $\mathbf{x}$ and $\mathbf{y}$ in high-dimensional space), and α_i is the Lagrangian coefficient corresponding to i-th training pair vector $\mathbf{p_i}$, obtained from the solution to the quadratic optimization problem. The output function $f(\mathbf{x})$ is bounded; the specific range of output values depends on the choice of kernel function K and the training set. For example, for radial basis function kernels $K(\mathbf{x}, \mathbf{x}_i) = \exp(-\frac{\|\mathbf{x}-\mathbf{x_i}\|^2}{2\sigma^2})$

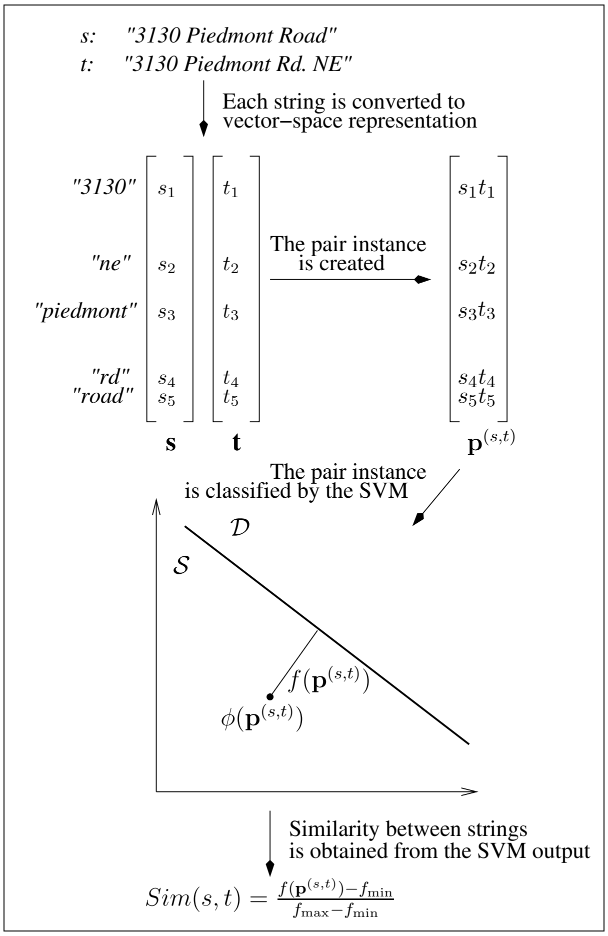

Figure 4: String similarity calculation for the SVM-based learnable vector-space model

(a very loose) bound is $|f(\mathbf{x})| \leq \sum_{i=1}^{l} \alpha_i + b$. Given upper and lower bounds $f_{\min} \leq f(\mathbf{x}) \leq f_{\max}$ for a particular kernel function and training set, it is trivial to obtain a similarity value in the $[0;1]$ range from the SVM output:

$$Sim_{VS-SVM}(s,t) = \frac{f(\mathbf{p}^{(s,t)}) - f_{\min}}{f_{\max} - f_{\min}} \tag{10}$$

The overall approach for obtaining a similarity value based on vector space using SVMs is shown in Figure 4, and the training algorithm is presented in Figure 5. To compare similarity between strings s and t, they are transformed to vector-space representations, and the pair vector $\mathbf{p}^{(\mathbf{s},\mathbf{t})}$ is formed from components of their vector representations. The pair vector is then classified by the trained SVM, and the final similarity value is obtained from the SVM prediction $f(\mathbf{p}^{(x,y)})$ using Equation (10).

3.3 Application of Learnable String Similarity Functions to Record Linkage

Accurate similarity functions are essential for identifying records that refer to the same object but are syntactically different. Two types of similarity functions are required for matching records: string distances for comparing individual field values and record distances for combining field similarities to obtain an overall

Input: Sets of equivalent strings $\mathcal{S} = \{(s,t),\ s \sim t\}$
and non-equivalent strings $\mathcal{D} = \{(s,t),\ s \nsim t\}$
Output: A string similarity function $Sim_{VS-SVM}(s,t)$
Method:
1. Convert all strings in $\mathcal{S}$ and $\mathcal{D}$ to vector representation
2. Create positive and negative training sets $\mathcal{S}_p$ and $\mathcal{D}_p$ by constructing pair instances $\mathbf{p}^{(s,t)} = \langle \frac{w_{v_i,s} w_{v_i,t}}{\|s\|\|t\|} \rangle$: $\mathcal{S}_p = \{\mathbf{p}^{(s,t)},\ (s,t) \in \mathcal{S}\}$; $\mathcal{D}_p = \{\mathbf{p}^{(s,t)},\ (s,t) \in \mathcal{D}\}$
3. Train an SVM on $\mathcal{S}_p$ and $\mathcal{D}_p$ to obtain the classification function $f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\mathbf{p_i}, \mathbf{x}) + b,\ f_{\min} \le f(\mathbf{x}) \le f_{\max}$
4. Return $Sim_{VS-SVM}(s,t) = \frac{f(\mathbf{p}^{(s,t)}) - f_{\min}}{f_{\max} - f_{\min}}$

Figure 5: Training algorithm for SVM-based vector space similarity

estimate of likeness between records. Thus, the record linkage problem permits evaluation of the learnable string similarity measures for two tasks: matching strings from individual field values and providing features for record-level similarity functions. Below, we describe our framework for using learnable string similarities for these two tasks and provide details of experiments on several record linkage datasets.

3.3.1 Learnable Similarity for Database Records

Because correspondence between overall record similarity and individual field similarities can vary greatly depending on how informative the fields are, an accurate record similarity function must weight fields according to their contribution to the true distance between records. While statistical aspects of combining similarity scores for individual fields have been addressed in previous work on record linkage (Winkler, 1999), availability of labeled duplicates allows a more direct approach that uses a binary classifier which computes a similarity function (Cohen & Richman, 2002). Given a database containing records composed of k different fields and a set $\mathcal{D} = \{D_1(\cdot,\cdot), \ldots, D_m(\cdot,\cdot)\}$ of similarity functions, we can represent any pair of records by an mk-dimensional vector. Each component of the vector represents similarity between two field values calculated using one of the m similarity functions.

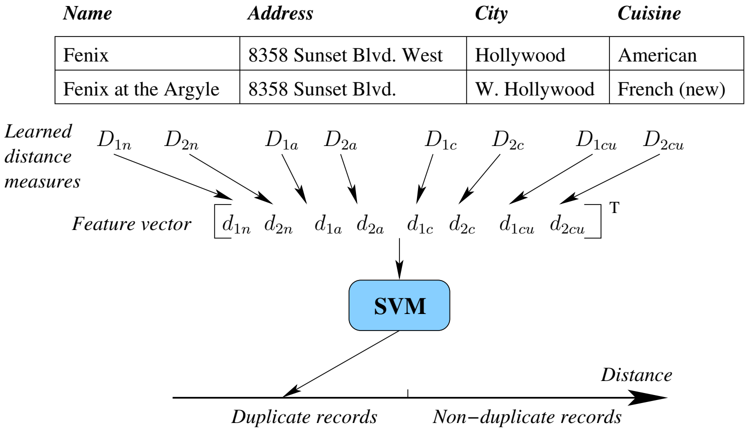

Figure 6: Computation of record similarity from individual field similarities

Matched pairs of equivalent records can be used to construct a training set of such feature vectors by assigning them a positive class label. Pairs of records that are not labeled as duplicates implicitly form a complementary set of negative examples. If the transitive closure of matched pairs contains disjoint sets of duplicate records, this approach will result in noisy negative examples. Then, a binary classifier is trained using these training vectors to discriminate between pairs of records corresponding to duplicates and non-duplicates. Overall, this approach follows the same framework that is used for learnable vector-space string similarity in the previous section. Following the same reasoning, support vector machines are an appropriate classifier choice due to their resilience to noise and their ability to handle correlated features well. The distance from the hyperplane provides a measure of confidence in the pair of records being a duplicate; it can be transformed to an actual similarity value using Equation (10). Figure 6 illustrates this process of computing record similarity using multiple similarity measures over each field and a binary classifier to categorize the resulting feature vector as belonging to the class of duplicates or non-duplicates. For each field of the database, two learnable distance measures, D_1 and D_2, are trained and used to compute similarity for that field. The values computed by these measures form the feature vector that is then classified by a support vector machine, producing a confidence value that represents similarity between the database records.

3.3.2 The Overall Record Linkage Framework

An overall view of our system, MARLIN, is presented in Figure 7. The training phase consists of two steps. First, the learnable string similarity functions are trained for each record field. The training corpus of paired field-level duplicates and non-duplicates is obtained by taking pairs of values for each field from the set of paired duplicate records. Because equivalent records may contain individual fields that are not equivalent, training data can be noisy. For example, if one record describing a restaurant contains *"Asian"* in the **Cuisine** field, and an equivalent record contains *"Seafood"*, a noisy training pair is formed that implies equivalence between these two strings. However, this issue does not pose a serious problem for our approach for two reasons. First, particularly noisy fields that are unhelpful for identifying record-level duplicates will be considered irrelevant by the classifier that combines similarities from different fields. Second, the presence of such pairs in the database indicates that there is a degree of similarity between such values, and using them in training allows the learnable record similarity function to capture that likelihood.

After individual string similarity functions are learned, they are used to compute distances for each field of training record pairs to obtain training data for the binary classifier in the form of vectors composed of distance features.

The record linkage phase starts with generation of potential duplicate pairs. Given a large database, producing all possible pairs of records and computing similarity between them is too expensive since it would require $O(n^2)$ distance computations. MARLIN utilizes the *canopies* clustering method (McCallum et al., 2000) using Jaccard similarity, a computationally inexpensive metric based on an inverted index, to separate records into overlapping clusters ("canopies") of potential equivalent records. Pairs of records that fall in each cluster then become candidates for a full similarity comparison shown in Figure 6.

Learned string similarity functions are then used to calculate distances for each field of each pair of potential equivalent records, forming field feature vectors for the classifier. Confidence estimates for belonging to the class of duplicates are produced by the binary classifier for each candidate pair, and pairs are sorted by increasing confidence.

The problem of finding a similarity threshold for separating duplicates from non-duplicates arises at this point. A trivial solution would be to use the binary classification results to label some records as duplicates, and others as non-duplicates. A traditional approach to this problem (Winkler, 1999), however, requires assigning two thresholds: one that separates pairs of records that are high-confidence duplicates, and another for possible duplicates that should be reviewed by a human expert. Since relative costs of labeling a non-duplicate as a duplicate (false positives) and overlooking true duplicates (false negatives)

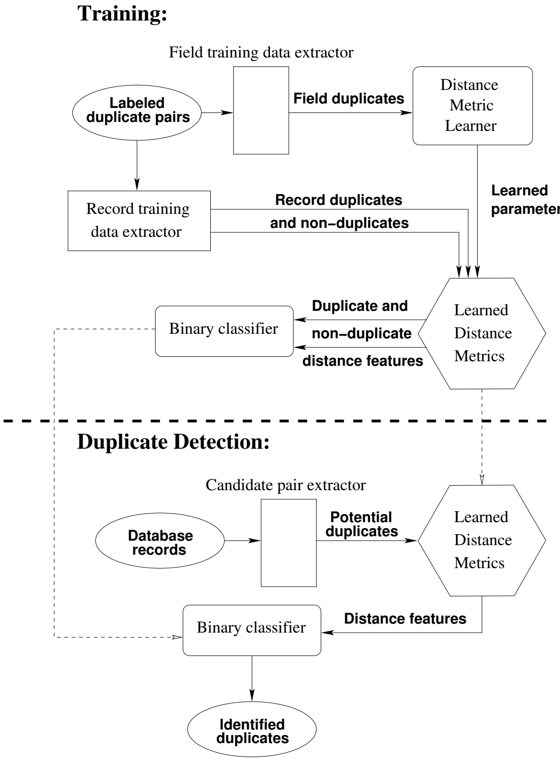

Figure 7: MARLIN overview

can vary from database to database, there is no "silver bullet" solution to this problem. Availability of labeled data, however, allows us to provide precision-recall estimates for any threshold value and thus offer a way to control the trade-off between false and unidentified duplicates by selecting threshold values that are appropriate for a particular database.

It is possible that several identified duplicate pairs will contain the same record. Since the "duplicate of" relation is transitive, it is necessary to compute the transitive closure of equivalent pairs to complete the identification process. Following (Monge & Elkan, 1997), MARLIN utilizes the union-find data structure to store all database records in this step, which allows updating the transitive closure of identified duplicates incrementally in an efficient manner.

3.3.3 Experimental Evaluation

Datasets. Our experiments were conducted on six datasets. *Restaurant* is a database of 864 restaurant names and addresses containing 112 duplicates obtained by integrating records from Fodor's and Zagat's guidebooks (Tejada et al., 2001). *Cora* is a collection of 1295 distinct citations to 122 Computer Science research papers from the Cora Computer Science research paper search engine (Cohen & Richman, 2002). The citations were segmented into multiple fields such as **author**, **title**, **venue**, etc. by an information extraction system, resulting in some crossover noise between the fields. *Reasoning*, *Face*, *Reinforcement* and *Constraint* are single-field datasets containing unsegmented citations to computer science papers in corresponding areas from the *Citeseer* scientific literature digital library[2] (Lawrence, Bollacker, & Giles,

[2]`http://citeseer.nj.nec.com/`

Table 1: Sample equivalent records from the *Cora* database

authors	**title**	**venue**	**address**	**year**	**pages**
Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby	Information, prediction, and query by committee	Advances in Neural Information Processing System	San Mateo, CA	1993	pages 483-490
Freund, Y., Seung, H. S., Shamir, E., & Tishby, N.	Information, prediction, and query by committee	Advances in Neural Information Processing Systems	San Mateo, CA.	–	(pp. 483-490).

Table 2: Sample equivalent records from the *Restaurant* database

name	**address**	**city**	**phone**	**cuisine**
Fenix	8358 Sunset Blvd. West	Hollywood	213/848-6677	American
Fenix at the Argyle	8358 Sunset Blvd.	W. Hollywood	213-848-6677	French(New)

Table 3: Sample equivalent records from the *Reasoning* database

L. P. Kaelbling. An architecture for intelligent reactive systems. In Reasoning About Actions and Plans: Proceedings of the 1986 Workshop. Morgan Kaufmann, 1986
Kaelbling, L. P., 1987. An architecture for intelligent reactive systems. In M. P. Georgeff & A. L. Lansky, eds., Reasoning about Actions and Plans, Morgan Kaufmann, Los Altos, CA, 395 410

1999). *Reasoning* contains 514 citation records that represent 196 unique papers, *Face* contains 349 citations to 242 papers, *Reinforcement* contains 406 citations to 148 papers, and *Constraint* contains 295 citations to 199 papers. Tables 1–3 contain sample duplicate records from the *Restaurant*, *Cora*, and *Reasoning* datasets.

Experimental Methodology. Every dataset was randomly split into 2 folds for cross-validation for each experimental run. A larger number of folds is impractical since it would result in few duplicate records per fold. To create the folds, duplicate records were grouped together, and the resulting clusters were randomly assigned to the folds, which resulted in uneven folds in some of the trials. All results are reported over 20 random splits, where for each split the two folds were used alternately for training and testing.

During each trial, record linkage was performed as described in Section 3.2. At each iteration, the pair of records with the highest similarity was labeled a duplicate, and the transitive closure of groups of duplicates was updated. Precision, recall and F-measure defined over pairs of duplicates were computed after each iteration, where precision is the fraction of identified duplicate pairs that are correct, recall is the fraction of actual duplicate pairs that were identified, and F-measure is the harmonic mean of precision and recall:

$$Precision = \frac{\#ofCorrectlyIdentifiedDuplicatePairs}{\#ofIdentifiedDuplicatePairs}$$

$$Recall = \frac{\#ofCorrectlyIdentifiedDuplicatePairs}{\#ofTrueDuplicatePairs}$$

$$F{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

As more pairs with lower similarity are labeled as duplicates, recall increases, and precision decreases because the number of non-duplicate pairs erroneously labeled as duplicates grows. Precision was interpo-

Table 4: Maximum F-measure for detecting equivalent field values

Similarity function	*Restaurant* **name**	*Restaurant* **address**	*Reasoning*	*Face*	*Reinforcement*	*Constraint*
Edit distance	0.290	0.686	0.927	0.952	0.893	0.924
Learned edit distance	**0.354**	**0.712**	**0.938**	**0.966**	**0.907**	**0.941**
Vector-space	0.365	0.380	0.897	**0.922**	**0.903**	0.923
Learned vector-space	**0.433**	**0.532**	**0.924**	0.875	0.808	0.913

lated at 20 standard recall levels following the traditional procedure in information retrieval (Baeza-Yates & Ribeiro-Neto, 1999).

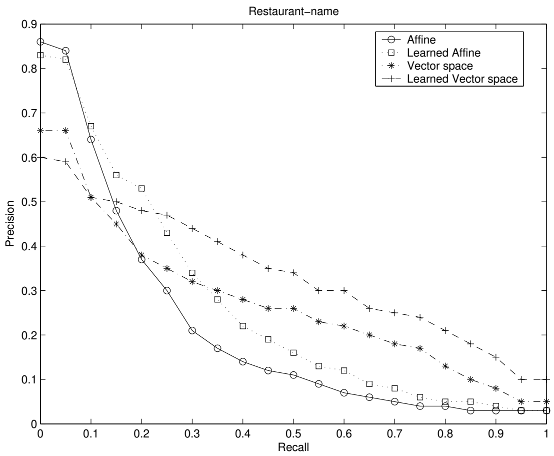

Figure 8: Field record linkage results for the **name** field of the *Restaurant* dataset.

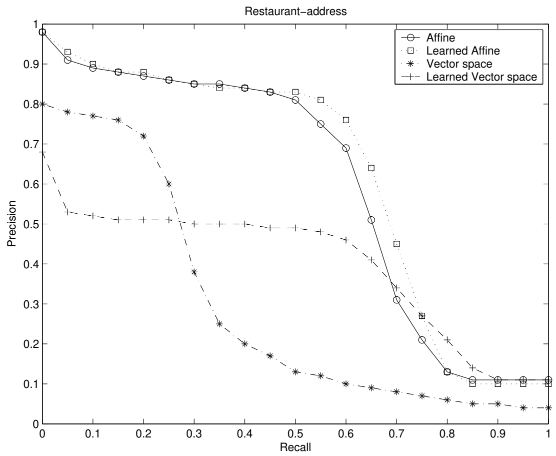

Figure 9: Field record linkage results for the **address** field of the *Restaurant* dataset.

Results: Field-level Record Linkage. To evaluate the performance of learnable string similarity functions, we compared learned string similarity functions with their fixed-cost analogs for the task of identifying equivalent field values. Along with the four single-field *Citeseer* datasets we chose two most meaningful fields from the *Restaurant* dataset - **name** and **address**.

We have compared four string similarity functions:

- Edit distance with affine gaps (Gusfield, 1997) using substitution cost of 2, gap opening cost of 3, gap extension cost of 1, and match cost of -1, which are commonly used parameters;
- Learnable edit distance with affine gaps described in Section 3.2.1, trained using the EM algorithm shown in Figure 3 with edit operation probabilities smoothed at $\lambda = 10^{-5}$;
- Normalized dot product in vector space (cosine similarity), computed using TF-IDF weights after stemming and stopword removal;
- Learnable vector-space SVM-based similarity described in Section 3.2.2, implemented over TF-IDF representation after stemming and stopword removal. SVM implementation based on the radial basis function kernel with width $\sigma = 10$ from the SVM^{light} package was used as the classifier (Joachims, 1999).

Results for field-level record linkage experiments are summarized in Table 4. Each entry in the table contains the average of maximum F-measure values over the 40 evaluated folds. The results in bold font indicate that the difference between the learned and the corresponding unlearned similarity function is significant at the 0.05 level using a 1-tailed paired t-test. Figures 8 and 9 contain recall-precision curves for the performance of MARLIN on the **name** and **address** fields of the *Restaurant* dataset respectively.

Relatively low precision of these two experiments is due to the fact that the duplicates for individual fields are very noisy due to falsely labeled negatives: for example, several restaurants from different cities may have variations of the same name, and in these trials these variations would be considered a non-duplicate. However, results in the following section will show that a combination of individual field estimates provides an accurate approximation of overall record similarities. The comparison of results for learned string similarity functions and the corresponding baselines shows that despite the noise, learned edit distance was able to adjust to the notion of similarity specific for each domain, while learned vector-space similarity improved over the standard vector-space similarity for approximately half of the domains.

It is peculiar that the learned vector space measure made more mistakes than unlearned cosine similarity for low recall values on the **name** and **address** datasets, but has outperformed it for higher recall. We conjecture that cosine similarity was able to correctly identify the few duplicates that were transpositions of exact matches (e.g. *"Hotel Bel-Air"* and *"Bel-Air Hotel"*, but has failed to give sufficiently high similarity to more difficult duplicates (e.g. *"Art's Deli"* and *"Art's Delicatessen"*). The SVM-trained string similarity function, on other hand, was able to generalize from the training examples, resulting in better similarity estimates across a range of more difficult duplicates while penalizing some of the "obvious" matches.

Overall, results of Table 4 show that learned affine edit distance outperforms both non-trained edit distance and vector-space cosine similarity for record linkage on individual fields. Visual inspection of the learned parameter values reveals that the parameters obtained by our training algorithm indeed capture certain domain properties that allow more accurate similarity computations. For example, for the **address** field of the *Restaurant* data, the lowest-cost edit operations include deleting *'e'* and deleting *'t'*, which capture the fact that a common cause of street name duplicates are abbreviations from *"Street"* to *"Str"*.

Results: Record-level Linkage. We evaluated the performance of MARLIN for multi-field record linkage. We again used the SVM^{light} implementation of a support vector machine as the binary classifier that combines similarity estimates across the different fields to produce the overall record similarity function as shown on Figure 6.

We have compared the performance of learnable and baseline string similarity functions for producing the similarity estimates of individual record fields. Table 5 summarizes the results for the *Restaurant* and *Cora* datasets. Again, results in bold font correspond to those experiments in which differences between the results for learned and corresponding unlearned string similarity functions are significant at the 0.05 level using a 1-tailed paired t-test. Figures 10 and 11 present the precision-recall curves for the experiments.

From these results we can conclude that using learnable string edit distance with affine gaps to compute similarity between field values makes a positive contribution when similarities from multiple fields are combined. Thus, better estimates of individual field similarities result in a more accurate calculation of the overall record similarity.

Using the SVM-based vector-space learnable similarity did not lead to improvements over the original vector space cosine similarity; performance has in fact decreased. Given that results for field-based record linkage with learnable vector-space similarity function were mixed, this is not surprising. We conducted additional experiments where the folds were created not by assigning the equivalence classes of duplicate records to folds, but by randomly assigning individual instances. This resulted in duplicate pairs correspond-

Table 5: Maximum F-measure for record linkage based on multiple fields

Similarity function	*Restaurant*	*Cora*
Edit distance	0.904	0.793
Learned edit distance	**0.922**	**0.824**
Vector space	**0.919**	**0.867**
Learned Vector space	0.826	0.803

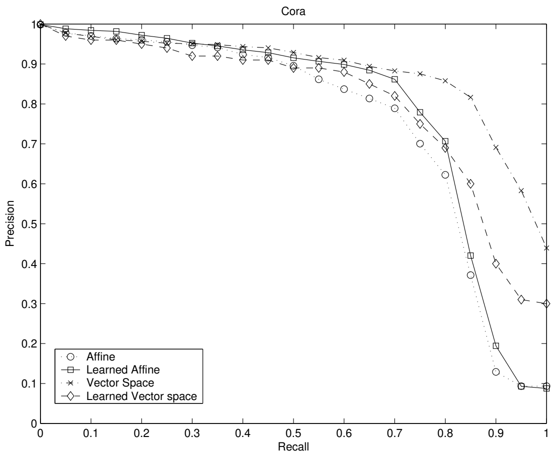

Figure 10: Record linkage results for the *Cora* dataset based on **author, title, venue, year** and **pages** fields

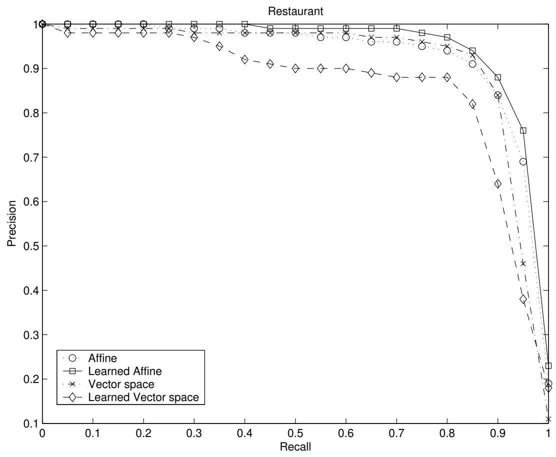

Figure 11: Record linkage results for the *Restaurant* dataset based on **name, address, city** and **cuisine** fields

ing to the same object being in both training and test sets (such experiments were only possible for the *Cora* and *Citeseer* datasets since there are at most 2 duplicates per equivalence class in the *Restaurant* dataset). In several of these experiments the learned vector-space similarity function outperformed the unlearned baseline. This can be explained by the fact that the training algorithm used by SVM^{light} relies on the assumption that the training data comes from the same distribution as the test data. Separating equivalence classes into different folds results in different token distributions for the two sets, and the learned classifier is not suitable for producing accurate predictions on the test data as a result. We believe that a number of enhancements to our algorithm can be made that will drastically improve on these initial results; we will describe them in detail in Section 4.1.1.

We also ran trials which combined sequence-based similarity functions (static and learnable string edit distance with affine gaps) and vector-space cosine similarity. These experiments resulted in near-100% precision and recall, without significant differences between static and adaptive field-level similarity functions. This demonstrates that combining sequence and token-based string similarity functions, such as learned edit distance with affine gaps and cosine similarity, is clearly an advantage of the two-level learning approach implemented in MARLIN. Current datasets did not allow us to show the benefits of adaptive similarity functions over their static prototypes in this scenario, but our preliminary results suggest that this can be demonstrated on more challenging datasets.

3.3.4 Training-Set Construction for Similarity Function Learning and Record Linkage

Training string and record similarity functions in real-world scenarios requires selecting a set of pairs for a human expert to label as duplicates or non-duplicates. Since typical corpora and databases contain few duplicates, selecting random pairs as potential training examples leads to training sets with extremely few identified duplicates (positive examples). As a result, such randomly selected training sets are highly skewed toward non-duplicates, which leads to suboptimal performance of similarity functions trained on this data. We propose two heuristic approaches for collecting training data: static-active learning and weakly-labeled selection, and then present experimental results on their effectiveness.

Static-active Selection of Training Pairs. Traditional active learning systems are "dynamic": labels of training examples selected in earlier rounds influence which unlabeled examples are deemed most informative in subsequent rounds. While prior work has examined dynamic active learning approaches to adaptive record linkage (Sarawagi & Bhamidipaty, 2002; Tejada et al., 2002), such strategies may not always be feasible due to high computational costs exacerbated by the large number of potential training examples. We propose using a "static" active learning method for selecting pairs of records that are [*likely* duplicates as a middle ground between computationally expensive dynamic active learning methods that try to identify the most informative training examples and random selection that is efficient but fails to select useful training data.

Our approach relies on the fact that off-the-shelf string similarity functions, such as the Jaro-Winkler metric or TF-IDF cosine similarity, can accurately identify duplicates at low recall levels (high confidence) even when duplicates are difficult to separate from non-duplicates at high recall levels (low confidence). Therefore, when a random sample of records from a database is taken and similarity between them is computed using such an off-the-shelf similarity function, string or record pairs with high similarity scores are likely to be duplicates. By asking the user to label strings or records with high textual similarity, a training sample with a high proportion of duplicates can be obtained. At the same time, non-duplicate records selected using this method are likely to be "near-miss" negative examples that are more informative for training than randomly selected record pairs most of which tend to be "obvious" non-duplicates.

Figures 12 and 13 demonstrate the comparative utility of static-active selection and random selection for choosing training record pairs on *Restaurant* and *Cora* datasets respectively. The record similarity function was trained on 40 training examples comprised of randomly selected record pairs and/or the most similar pairs selected by a static-active method using TF-IDF cosine similarity. Using a token-based inverted index for the vector-space model (Baeza-Yates & Ribeiro-Neto, 1999) allowed efficient selection of static-active training examples without computing similarity between all pairs of records. All experiments utilized SVM^{light} for computing learnable record similarity function and two unlearned string similarity functions for field comparisons: TF-IDF cosine similarity and edit distance with affine gaps.

For both datasets, the highest performance is achieved when record similarity functions are trained using a mix of static-active and randomly selected pairs. However, employing many random pairs with a few static-active examples yields the best results on *Cora*, while on *Restaurant* the highest performance is achieved when the system is trained on a balanced mix of static-active and random examples. This difference is explained by the makeup of the two datasets. *Cora* has a higher absolute number of duplicates than *Restaurant* (8592 versus 56 for each fold); duplicates in *Cora* also represent a larger proportion of all record pairs (8592/211242 versus 56/93406 for each fold). On *Restaurant*, random selection results in training sets that contain almost no duplicates, while including a significant number of pairs selected using the static-active technique leads to balanced training sets that contain sufficient positive and negative examples. On *Cora*, however, randomly selected record pairs are likely to contain a few duplicates. Including a limited

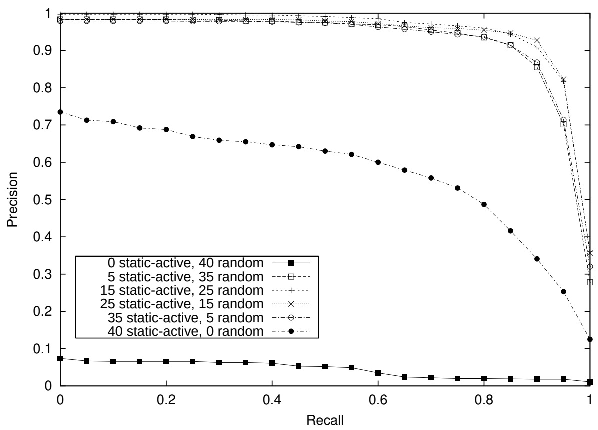

Figure 12: Comparison of random and static-active training example selection on the *Restaurant* dataset.

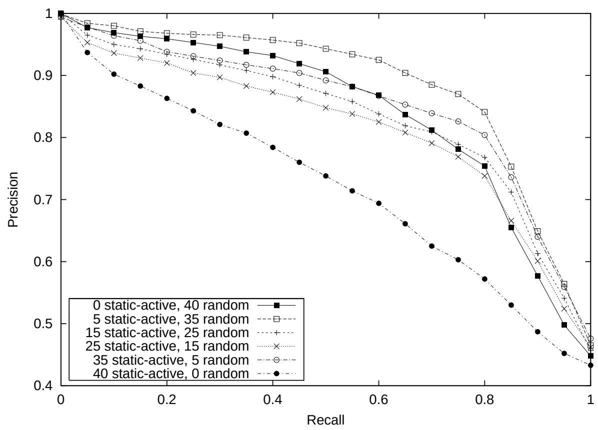

Figure 13: Comparison of random and static-active training example selection on the *Cora* dataset.

number of record pairs chosen using the static-active technique results in the best performance, but as more static-active examples are added, performance decreases because highly similar duplicates take the place of informative non-duplicates in the training set. Thus, the worst performance on *Restaurant* occurs when all training examples are chosen randomly because duplicates are almost never encountered, while on *Cora* using only examples chosen by static-active selection results in the opposite problem: extremely few non-duplicate pairs are found, and the class distribution of training data is highly skewed toward duplicates.

Based on these results, we conclude that best training sets for learnable record similarity functions are obtained when randomly chosen pairs of records are combined with pairs chosen using static-active selection. The specific proportion in which the two kinds of training data should be mixed can be estimated based on the outcome of labeling randomly chosen pairs. If duplicates are exceptionally rare, a significant number of static-active examples is required to obtain a sufficient sample of duplicates, while databases with a large number of duplicates need only a small number of record pairs selected using the static-active methodology to complete a representative training set.

Overall, we show that a reasonable baseline to which dynamic active learning methods for adaptive similarity functions should be compared is not the one that uses only randomly selected training pairs, but one that employs the static-active method to overcome the extreme skewness in class distribution that is typical for similarity function learning and record linkage problems.

Weakly-labeled Selection. While the static-active method allows identifying duplicate training pairs for learnable similarity functions, the inverse problem can be encountered in some real-world situations: a "legacy" training set consisting of identified duplicates may be available, while informative non-duplicate pairs need to be collected. For such situations we consider an unsupervised technique for obtaining negative examples. Since duplicate records are rare in a typical database, two randomly selected records are likely to be non-duplicates, and therefore can potentially be used as negative training examples for learning similarity functions. To help ensure that no duplicate records are included among these pairs, only pairs of records that do *not* share a significant number of common tokens should be included as negative examples. Such selection of "weakly-labeled" (and potentially noisy) non-duplicate record pairs is the unsupervised analog

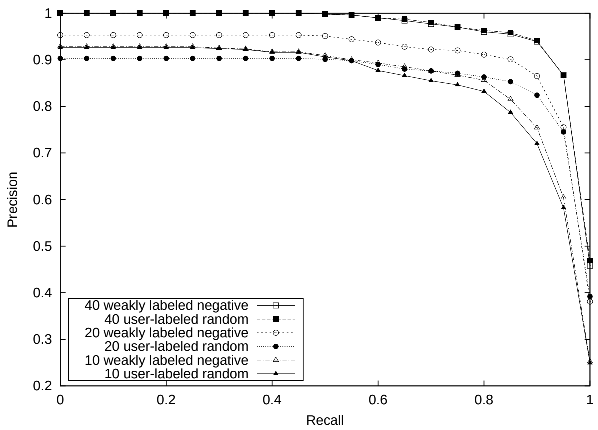

Figure 14: Comparison of using weakly-labeled non-duplicates with using random labeled record pairs on the *Restaurant* dataset.

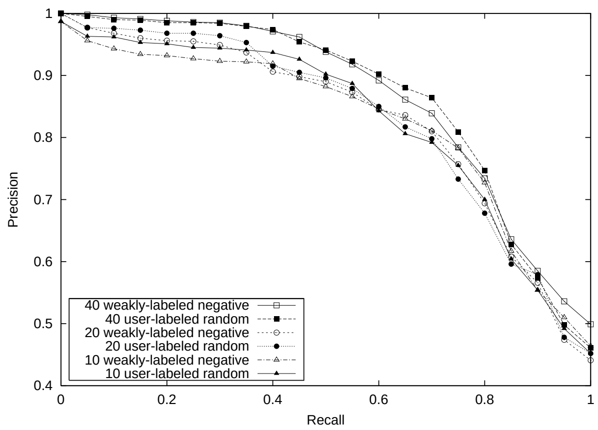

Figure 15: Comparison of using weakly-labeled non-duplicates with using random labeled record pairs on the *Cora* dataset.

of static-active selection of duplicates. The process can also be thought of as the opposite of blocking or canopies techniques that use off-the-shelf metrics to avoid comparing "obvious" non-duplicates to speed up the record linkage process.

We compared the record linkage accuracy of MARLIN trained on weakly-labeled negatives with training on user-labeled negatives. Figures 14 and 15 present the results of these experiments on the *Restaurant* and *Cora* datasets. Weakly-labeled negatives were selected randomly from record pairs that shared no more than 20% of tokens to minimize the noise. All experiments used training sets composed of two parts: half the examples were positives randomly selected among user-labeled duplicate pairs, and the other half was composed of either weakly-labeled non-duplicate records or randomly selected labeled record pairs. SVM^{light} was employed to compute record similarity, and TF-IDF cosine similarity and edit distance with affine gaps were used as the underlying string similarity functions for individual fields.

The results again demonstrate that the utility of heuristic selection of training data for similarity function learning is dataset-dependent. On *Restaurant*, where duplicate pairs are scarce and randomly selected records are true non-duplicates with very high probability, using weakly-labeled non-duplicates yields results identical to randomly selected labeled duplicates when a large number of examples is selected, and improves slightly over random selection when the training set is small. We conjecture that biasing the SVM with "negative but slightly similar" examples when very little training data is available allows learning a better separating hyperplane. On *Cora*, using weakly-labeled negatives leads to slight degradation of system accuracy, which is expected since duplicates are relatively frequent, and noise is likely to be introduced when negative examples are collected in an unsupervised manner. However, the drop in performance is small, and in situations where human labeling of negatives is expensive or infeasible (e.g. due to privacy issues), using weakly-labeled selection is a viable avenue for unsupervised acquisition of negative training examples for similarity function learning.

3.4 MPC-KMEANS: Combining Similarity Function Learning and Seeding in K-Means

In previous section, we have shown that using learnable string similarity functions leads to improvements in record linkage accuracy. Clustering is another application that could benefit from using adaptive metrics, and in this section we describe MPC-KMEANS, our method for integrating an adaptive variant of Euclidean distance in a constrained clustering algorithm.

Semi-supervised Clustering with Constraints In a *semi-supervised clustering* setting, a small amount of labeled data is available to aid the unsupervised clustering process. For pairwise constrained clustering, we consider a framework that has pairwise must-link and cannot-link constraints (with an associated cost of violating each constraint) between points in a dataset, in addition to having distances between the points (Wagstaff et al., 2001). Supervision in the form of constraints is generally more practical than providing class labels in the clustering framework, since true labels may be unknown a priori, while a human expert can easily specify whether pairs of points belong to the same cluster or different clusters.

Since K-Means clustering cannot handle pairwise constraints explicitly, we formulate the goal of clustering in the pairwise constrained clustering framework as minimizing a combined objective function, which is defined as the sum of the total square distances between the points and their cluster centroids and the cost of violating the pairwise constraints. The mathematical formulation of this framework is motivated by the *metric labeling* problem and the *generalized Potts* model (Kleinberg & Tardos, 1999; Boykov, Veksler, & Zabih, 1998).

In the pairwise constrained clustering framework, let $\mathcal{M}$ be the set of unordered must-link pairs such that $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ implies $\mathbf{x}_i$ and $\mathbf{x}_j$ should be assigned to the same cluster, and $\mathcal{C}$ be the set of unordered cannot-link pairs such that $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ implies $\mathbf{x}_i$ and $\mathbf{x}_j$ should be assigned to different clusters. Let $W = \{w_{ij}\}$ and $\overline{W} = \{\overline{w}_{ij}\}$ be two sets that give the weights corresponding to the must-link constraints in $\mathcal{M}$ and the cannot-link constraints in $\mathcal{C}$ respectively. Let d_M and d_C be two metrics that quantify the cost of violating must-link and cannot-link constraints: $d_M(l_i, l_j) = \mathbb{1}[l_i \neq l_j]$ and $d_C(l_i, l_j) = \mathbb{1}[l_i = l_j]$, where $\mathbb{1}$ is the indicator function ($\mathbb{1}[true]$ = 1, $\mathbb{1}[false]$ = 0), and l_i and l_j are the cluster labels for $\mathbf{x}_i$ and $\mathbf{x}_j$. Using this model, the problem of pairwise constrained clustering under must-link and cannot-link constraints is formulated as minimizing the following objective function, where point $\mathbf{x}_i$ is assigned to the partition $\mathcal{X}_{l_i}$ with centroid $\boldsymbol{\mu}_{l_i}$:

$$\mathcal{J}_{\text{pckm}} = \sum_{\mathbf{x}_i \in \mathcal{X}} \|\mathbf{x}_i - \boldsymbol{\mu}_{l_i}\|^2 + \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}} w_{ij} \mathbb{1}[l_i \neq l_j] + \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}} \overline{w}_{ij} \mathbb{1}[l_i = l_j] \tag{11}$$

We will refer to this model as the pairwise constrained K-Means (PC-KMEANS) model.

Semi-supervised Clustering via Similarity Function Learning Another avenue for utilizing labeled data involves adapting the similarity function employed by the clustering algorithm. Intuitively, this allows capturing the user's view of which objects should be considered similar and which dissimilar. Since the original data representation may not be embedded in a space where clusters are sufficiently separated, modifying the similarity function transforms the representation so that distances between same-cluster objects are minimized, while distances between different-cluster objects are maximized. As a result, clusters discovered using the learned distance functions adhere more closely to the notion of similarity expressed by the labeled data than the clusters obtained using untrained similarity functions.

Following previous work (Xing et al., 2003), we can parameterize Euclidean distance with a symmetric positive-definite weight matrix $\mathbf{A}$ as follows: $\|\mathbf{x}_i - \mathbf{x}_j\|_{\mathbf{A}} = \sqrt{(\mathbf{x}_i - \boldsymbol{\mu}_{l_i})^T \mathbf{A} (\mathbf{x}_i - \boldsymbol{\mu}_{l_i})}$. If $\mathbf{A}$ is restricted to be a diagonal matrix, then it scales each axis by a different weight and corresponds to feature weighting; otherwise new features are created that are linear combinations of the original features. In our clustering formulation, using the matrix $\mathbf{A}$ is equivalent to considering a generalized version of the K-Means model described in Section 2.6, where all the Gaussians have a covariance matrix $\mathbf{A}^{-1}$ (Bilmes, 1997).

It can be easily shown that maximizing the complete data log-likelihood under this generalized K-Means model with metric learning is equivalent to minimizing the objective function:

$$\mathcal{J}_{\text{mkmeans}} = \sum_{\mathbf{x}_i \in \mathcal{X}} \|\mathbf{x}_i - \boldsymbol{\mu}_{l_i}\|^2_{\mathbf{A}} - \log(\det(\mathbf{A})) \tag{12}$$

where the second term arises due to the normalizing constant of a Gaussian with covariance matrix $\mathbf{A}^{-1}$.

Unifying Constraints and Similarity Function Learning in Clustering Previous work on semi-supervised clustering (Cohn et al., 2000; Xing et al., 2003) that used labeled data for learning a similarity function only utilized pairwise constraint information to learn weights that minimize constraint violations. We incorporate similarity function learning directly into the clustering algorithm in a way that allows unlabeled data to influence the similarity function learning process along with pairwise constraints. Combining objective functions (11) and (12) leads to the following objective function that attempts to minimize cluster dispersion under a learned metric along with minimizing the number of constraint violations:

$$\mathcal{J}_{\text{combined}} = \sum_{\mathbf{x}_i \in \mathcal{X}} \|\mathbf{x}_i - \boldsymbol{\mu}_{l_i}\|^2_{\mathbf{A}} + \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{M}} w_{ij} \mathbb{1}[l_i \neq l_j] + \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{C}} \overline{w}_{ij} \mathbb{1}[l_i = l_j] - \log(\det(\mathbf{A})) \tag{13}$$

If we assume uniform weights w_{ij} and $\overline{w}_{ij}$, as traditionally done in the generalized Potts model (Boykov et al., 1998), one problem with this objective function would be that all constraint violations are treated equally. However, the cost of violating a must-link constraint between two *close* points should be higher than the cost of violating a must-link constraint between two points that are *far apart*. Such cost assignment reflects the intuition that it is a worse error to violate a must-link constraint between similar points, and such an error should have more impact on the similarity function learning framework. Multiplying the weights w_{ij} with the penalty function $f_M(\mathbf{x}_i, \mathbf{x}_j) = \max(\alpha_{\min}, \alpha_{\max} - \|\mathbf{x}_i - \mathbf{x}_j\|^2_{\mathbf{A}})$ gives us the overall cost of violating a must-link constraint between two points $\mathbf{x}_i$ and $\mathbf{x}_j$, where $\alpha_{\min}$ and $\alpha_{\max}$ are non-negative constants that correspond to minimum and maximum penalties respectively. They can be set as fractions of the square of the maximum must-link distance $\max_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} \|\mathbf{x}_i - \mathbf{x}_j\|^2$, thus guaranteeing that the penalty for violating a constraint is always positive. Overall, this formulation enables the penalty for violating a must-link constraint to be proportional to the seriousness of the violation.

Analogously, the cost of violating a cannot-link constraint between two *dissimilar* points should be higher than the cost of violating a cannot-link constraint between points that are *similar*, since the former is a "worse error". Multiplying weights $\overline{w}_{ij}$ with $f_C(\mathbf{x}_i, \mathbf{x}_j) = \min(\alpha_{\min} + \|\mathbf{x}_i - \mathbf{x}_j\|^2_{\mathbf{A}}, \alpha_{\max})$ allows us to take the "seriousness" of the constraint violation into account. The combined objective function then becomes:

$$\begin{aligned}\mathcal{J}_{\text{mpc-kmeans}} = &\sum_{\mathbf{x}_i \in \mathcal{X}} \|\mathbf{x}_i - \boldsymbol{\mu}_{l_i}\|^2_{\mathbf{A}} - \log(\det(\mathbf{A})) \\ &+ \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{M}} w_{ij} f_M(\mathbf{x}_i, \mathbf{x}_j) \mathbb{1}[l_i \neq l_j] + \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{C}} \overline{w}_{ij} f_C(\mathbf{x}_i, \mathbf{x}_j) \mathbb{1}[l_i = l_j]\end{aligned} \tag{14}$$

The weights w_{ij} and $\overline{w}_{ij}$ provide a way to specify the relative importance of the unlabeled versus labeled data while allowing individual constraint weights. This objective function $\mathcal{J}_{\text{mpc-kmeans}}$ is greedily optimized by our proposed metric pairwise constrained K-Means (MPC-KMEANS) algorithm that uses a K-Means-type iteration.

Given a set of data points $\mathcal{X}$, a set of must-link constraints $\mathcal{M}$, a set of cannot-link constraints $\mathcal{C}$, corresponding weight sets W and $\overline{W}$, and the number of clusters to form K, metric pairwise constrained K-Means (MPC-KMEANS) finds a disjoint K partitioning $\{\mathcal{X}_h\}_{h=1}^{K}$ of $\mathcal{X}$ (with each partition having a centroid $\boldsymbol{\mu}_h$) such that $\mathcal{J}_{\text{mpc-kmeans}}$ is (locally) minimized.

The algorithm MPC-KMEANS has two components. Utilizing constraints during cluster initialization and satisfaction of the constraints during every cluster assignment step constitutes the search-based component of the algorithm. Learning the similarity function by re-estimating the weight matrix $\mathbf{A}$ during each algorithm iteration based on current constraint violations is the similarity-based component.

Intuitively, the search-based technique uses the pairwise constraints to generate seed clusters that initialize the clustering algorithm, and also uses the constraints to guide the clustering process through the iterations. At the same time, the similarity-based technique distorts the metric space to minimize the costs of violated constraints, possibly removing the violations in the subsequent iterations. Implicitly, the space where data points are embedded is transformed to respect the user-provided constraints, thus learning a similarity function that is appropriate for the dataset from the user's perspective.

The Algorithm. MPC-KMEANS starts by generating the seed clusters in the initialization step. A set of λ neighborhood sets $\{N_p\}_{p=1}^{\lambda}$ is created by taking the transitive closure of the must-link constraints, augmenting the set of must-link constraints $\mathcal{M}$ by adding these entailed constraints, and using the resulting connected components to create the neighborhood sets. The set of cannot-link constraints $\mathcal{C}$ is augmented in a similar manner. The resulting set of neighborhoods provides a good initial starting point for the MPC-KMEANS algorithm. The initial cluster centers are initialized using k neighborhood sets of largest size.

Then, MPC-KMEANS alternates between cluster assignments in the Expectation step and centroid estimation and similarity function learning in the Maximization step until convergence. In the E-step, every point $\mathbf{x}$ is assigned to a cluster so that the sum of the distance of $\mathbf{x}$ to the cluster centroid and the cost of constraint violations possibly incurred by this cluster assignment is minimized. Each point moves to a new cluster only if the component of $\mathcal{J}_{\text{mpc-kmeans}}$ contributed by this point decreases, so when all points are given their new assignment, $\mathcal{J}_{\text{mpc-kmeans}}$ will decrease or remain the same.

In the M-step, the cluster centroids $\boldsymbol{\mu}_h$ are first re-estimated using the points in $\mathcal{X}_h$. As a result, the contribution of each cluster to $\mathcal{J}_{\text{mpc-kmeans}}$ via its first term is minimized. Then, similarity function learning is performed: the matrix $\mathbf{A}$ is re-estimated to decrease the objective function $\mathcal{J}_{\text{mpc-kmeans}}$. The updated matrix $\mathbf{A}$ is obtained by taking the partial derivative $\frac{\partial \mathcal{J}_{\text{mpc-kmeans}}}{\partial \mathbf{A}}$ and setting it to zero, resulting in:

$$
\begin{aligned}
\mathbf{A} = \; & \Bigg(\sum_{\mathbf{x}_i \in \mathcal{X}} (\mathbf{x}_i - \boldsymbol{\mu}_{l_i})(\mathbf{x}_i - \boldsymbol{\mu}_{l_i})^T \\
& - \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{M}^*} w_{ij} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \mathbb{1}[l_i \neq l_j] + \sum_{(\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{C}^*} \overline{w}_{ij} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \mathbb{1}[l_i = l_j] \Bigg)^{-1}
\end{aligned}
\tag{15}
$$

where M^* and C^* are subsets of M and C that exclude the constraint pairs for which the penalty functions f_M and f_C take the threshold values α_{min} and α_{max} respectively.

Since estimating a full matrix $\mathbf{A}$ from limited training data is difficult, we currently limit ourselves to diagonal $\mathbf{A}$, which is equivalent to similarity function learning via feature weighting. In that case, the d-th diagonal element of $\mathbf{A}$, a_{dd}, corresponds to the weight of the d-th feature:

$$a_{dd} = \Bigg(\sum_{\mathbf{x}_i \in \mathcal{X}} (\mathbf{x}_{id} - \boldsymbol{\mu}_{l_i d})^2 - \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}^*} w_{ij} (\mathbf{x}_{id} - \mathbf{x}_{jd})^2 \mathbb{1}[l_i \neq l_j] + \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}^*} \overline{w}_{ij} (\mathbf{x}_{id} - \mathbf{x}_{jd})^2 \mathbb{1}[l_i = l_j] \Bigg)^{-1}$$

Intuitively, the first term in the sum, $\sum_{\mathbf{x}_i \in \mathcal{X}} (\mathbf{x}_{id} - \boldsymbol{\mu}_{l_i d})^2$, scales the weight of each feature proportionately to the feature's contribution to the overall cluster dispersion, analogously to scaling performed when computing Mahalanobis distance. The last two terms,
$-\sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}^*} w_{ij} (\mathbf{x}_{id} - \mathbf{x}_{jd})^2 \mathbb{1}[l_i \neq l_j]$ and $\sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}^*} \overline{w}_{ij} (\mathbf{x}_{id} - \mathbf{x}_{jd})^2 \mathbb{1}[l_i = l_j]$, handle constraint violations by respectively contracting and stretching each dimension to reduce the current violations. Thus, the distance function weights are adjusted at each iteration in such a way that the contribution of different attributes to similarity is equalized, while constraint violations are minimized.

Algorithm: `m-PCKMeans`
Input: Set of data points $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$,
set of *must-link* constraints $\mathcal{M} = \{(\mathbf{x}_i, \mathbf{x}_j)\}$,
set of *cannot-link* constraints $\mathcal{C} = \{(\mathbf{x}_i, \mathbf{x}_j)\}$,
number of clusters K, sets of constraint weights W and $\overline{W}$.
Output: Disjoint K partitioning $\{\mathcal{X}_h\}_{h=1}^K$ of $\mathcal{X}$ such that
objective function $\mathcal{J}_{\text{mpc-kmeans}}$ is (locally) minimized.
Method:
1. Initialize clusters:
1a. create the λ neighborhoods $\{N_p\}_{p=1}^{\lambda}$ from $\mathcal{M}$ and $\mathcal{C}$
1b. sort the indices p in decreasing size of N_p
1c. if $\lambda \geq K$
initialize $\{\boldsymbol{\mu}_h^{(0)}\}_{h=1}^K$ with centroids of $\{N_p\}_{p=1}^K$
else if $\lambda < K$
initialize $\{\boldsymbol{\mu}_h^{(0)}\}_{h=1}^{\lambda}$ with centroids of $\{N_p\}_{p=1}^{\lambda}$
if $\exists$ point $\mathbf{x}$ *cannot-linked* to all neighborhoods $\{N_p\}_{p=1}^{\lambda}$
initialize $\boldsymbol{\mu}_{\lambda+1}^{(0)}$ with $\mathbf{x}$
initialize remaining clusters at random
2. Repeat until *convergence*
2a. `assign_cluster`: Assign each data point $\mathbf{x}_i$ to cluster h^*
(i.e. set $\mathcal{X}_{h^*}^{(t+1)}$), for $h^* = \arg\min_h (\|\mathbf{x}_i - \boldsymbol{\mu}_h^{(t)}\|_{\mathbf{A}}^2 - \log(\det(\mathbf{A}))$
$+ \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}} w_{ij} f_M(\mathbf{x}_i, \mathbf{x}_j) \mathbb{1}[h \neq l_j]$
$+ \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}} \overline{w}_{ij} f_C(\mathbf{x}_i, \mathbf{x}_j) \mathbb{1}[h = l_j])$
2b. `estimate_means`: $\{\boldsymbol{\mu}_h^{(t+1)}\}_{h=1}^K \leftarrow \{\frac{1}{|\mathcal{X}_h^{(t+1)}|} \sum_{\mathbf{x} \in \mathcal{X}_h^{(t+1)}} \mathbf{x}\}_{h=1}^K$
2c. `update_metric`: $\mathbf{A}^{-1} = \sum_{\mathbf{x}_i \in \mathcal{X}} (\mathbf{x}_i - \boldsymbol{\mu}_{l_i})(\mathbf{x}_i - \boldsymbol{\mu}_{l_i})^T$
$- \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}^*} w_{ij} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \mathbb{1}[l_i \neq l_j]$
$+ \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}^*} \overline{w}_{ij} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T \mathbb{1}[l_i = l_j]$
2d. $t \leftarrow (t + 1)$

Figure 16: MPC-KMEANS algorithm

The objective function decreases after every cluster assignment, centroid re-estimation and similarity function learning step till convergence, implying that the MPC-KMEANS algorithm will converge to a local minima of $\mathcal{J}_{\text{mpc-kmeans}}$.

Experiments: Methodology and Datasets Experiments were conducted on several datasets from the UCI repository: *Iris*, *Wine*, and representative randomly sampled subsets from the *Pen-Digits* and *Letter* datasets. For *Pen-Digits* and *Letter*, we chose two sets of three classes: $\{$**I, J, L**$\}$ from *Letter* and $\{$**3, 8, 9**$\}$ from *Pen-*

Digits, sampling 20% of the data points from the original datasets randomly. These classes were chosen from the handwriting recognition datasets since they intuitively represent difficult visual discrimination problems.

We used pairwise F-measure to evaluate the clustering results similarly to record linkage result evaluation described above. Such evaluation is based on the traditional information retrieval measures, adapted for evaluating clustering by considering same-cluster pairs:

$$Precision = \frac{\#PairsCorrectlyPredictedInSameCluster}{TotalPairsPredictedInSameCluster}$$

$$Recall = \frac{\#PairsCorrectlyPredictedInSameCluster}{TotalPairsInSameCluster}$$

$$F-measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

We generated learning curves with 10-fold cross-validation for each dataset to determine the effect of utilizing the pairwise constraints. Each point in the learning curve represents a particular number of pairwise constraints given as input to the algorithm. Unit constraint weights W and $\overline{W}$ were used, since the datasets did not provide individual weights for the constraints. The maximum square distance between must-link constraints was used as value for $\alpha_{\max}$, while $\alpha_{\min}$ was set to 0. The clustering algorithm was run on the whole dataset, but the pairwise F-measure was calculated only on the test set. Results were averaged over 50 runs of 10 folds.

Experiments: Results and Discussion Figures 17-20 show learning curves for the four datasets. For each dataset, we compared four semi-supervised clustering schemes:

- MPC-KMEANS clustering, which involves both seeding and similarity function learning in the unified framework described above;
- M-KMEANS, which is K-Means clustering with the learnable similarity function component described above, without utilizing constraints for seeding;
- PC-KMEANS clustering, which utilizes constraints for seeding the initial clusters and forces the cluster assignments to respect the constraints without doing any similarity function learning, as outlined above;
- Unsupervised K-Means clustering.

The learning curves illustrate that providing pairwise constraints is beneficial to clustering quality: as the amount of supervision increases, PC-KMEANS, M-KMEANS, and MPC-KMEANS produce clusters that are more homogeneous with respect to the data classes, which is indicated by rising learning curves. Overall, the unified approach (MPC-KMEANS) outperforms individual seeding (PC-KMEANS) and similarity function learning (M-KMEANS) approaches on the presented datasets.

For the *Wine* and *Letter-IJL* datasets, the difference at 0 pairwise constraints between methods that utilize similarity function learning (MPC-KMEANS and M-KMEANS) and those that do not (PC-KMEANS and regular K-Means) indicates that even in the absence of constraints, weighting features by their variance (essentially using Mahalanobis distance) improves clustering accuracy. For the *Wine* dataset, additional constraints provide an insubstantial improvement in cluster quality on this dataset, which shows that meaningful feature weights are obtained from scaling by variance using just the unlabeled data.

Some of the similarity function learning curves display a characteristic "dip", where clustering accuracy decreases when initial constraints are provided, but after a certain point starts to increase and eventually

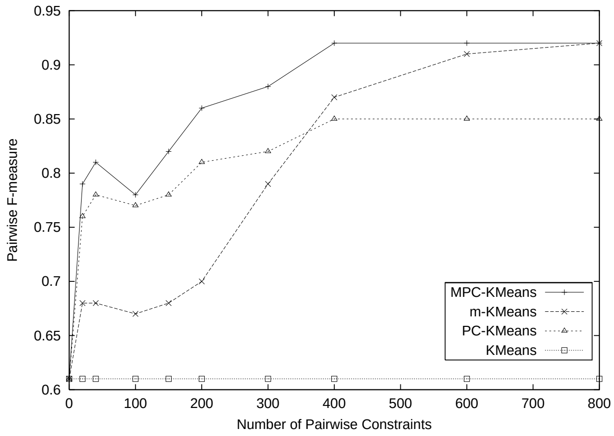

Figure 17: Results on the *Iris* dataset

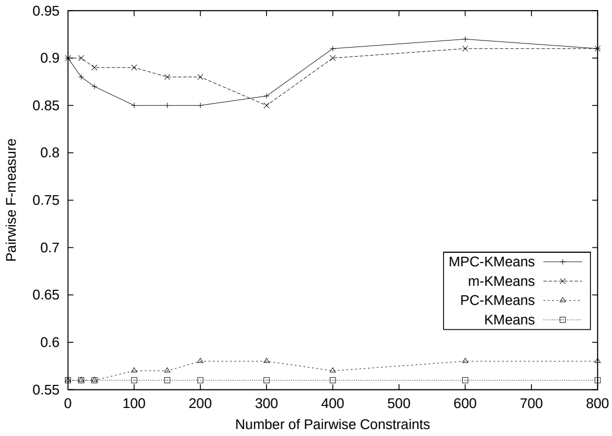

Figure 18: Results on the *Wine* dataset

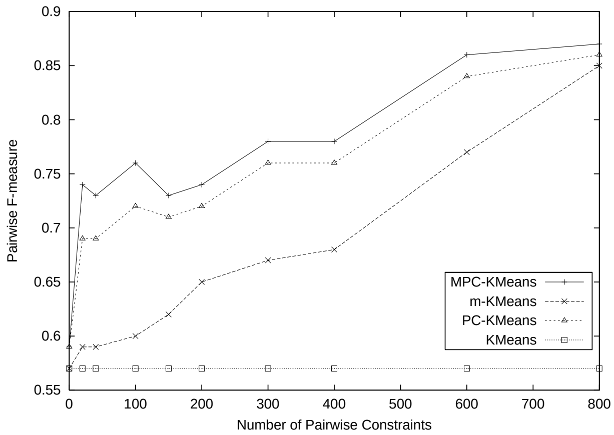

Figure 19: Results on the *Digits-389* dataset

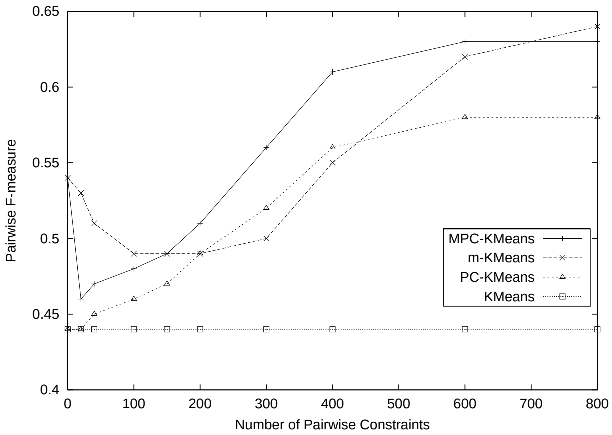

Figure 20: Results on the *Letter-IJL* dataset

outperforms the initial point of the learning curve. We conjecture that this phenomenon is due to the fact that feature weights learned from few constraints are unreliable, while increasing the number of constraints provides the similarity function learning mechanism enough data to estimate good parameter values.

On the other hand, seeding the clusters with a small number of pairwise constraints has an immediate positive effect on the final cluster quality, while providing more pairwise constraints has diminishing returns, i.e., PC-KMEANS learning curves rise slowly. When both seeding and similarity function learning are utilized, the unified approach benefits from the individual strengths of the two methods, as can be seen from the MPC-KMEANS results.

Overall, our results indicate that combining similarity function learning with seeding in clustering outperforms these methods in isolation and leads to improvements in cluster quality.

4 Proposed Work

4.1 Learnable String Similarity Functions

Improvements obtained using learnable string edit distance with affine gaps and learnable vector-space string similarity indicate that developing methods for adapting string metrics to particular domains is a promising area for future research. In proposed work, we plan to extend the methods that we described in Section 3.2, as well as develop several new techniques for training learnable similarity functions.

4.1.1 Learnable Vector-space String Similarity

Improving Vector-space Similarity Based on Pairwise Classification. In Section 3.2.2 we described a general approach for learning vector-space similarity using a binary classifier. While our initial experiments produced mixed results, we intend to add several enhancements to this technique that should lead to significant improvements.

In experiments described in Section 3.3.4, the hyperplanes produced by the SVM typically rely on a large number of support vectors (which are training pairs of similar and dissimilar strings). This indicates that the learned separating hyperplane has high bias, which predictably leads to high error (Domingos, 2000; Valentini & Dietterich, 2003). However, simply altering the SVM parameters to lower bias by decreasing the number of support vectors is a flawed strategy. Since we focus on similarity for domains where strings are short, their vector representations are highly sparse, which leads to very few words being selected as classification features. If the number of support vectors is reduced artificially by changing kernel parameters, even fewer words are selected as discriminating features, which brings us back to increased bias and high error. We plan to investigate the following strategies that can solve this problem:

- Because using word tokenization for obtaining the vector-space representation of strings leads to very sparse vectors, the pairwise classification approach to learnable string similarity would benefit from a more expressive string representation. In recent work on string kernels for text classification, Lodhi et al. (2002) proposed mapping strings into the feature space generated by all subsequences of fixed length. Instead of word or n-gram tokenization, strings are represented by features corresponding to all of their ordered fixed-length subsequences that are not necessarily contiguous. For example, the following length-3 features represent the string *'8 str'*: {*'8_s', '_st', 'str', '8_t', '8_r', '8st', '8sr', '8tr', '_sr', '_tr'*}. Feature weights for non-contiguous subsequences are penalized by using a decay factor that scales weights proportionally to the subsequence length in the original string.

  Explicit computation of such vector-space representation is inefficient for large text documents since it would lead to very dense high-dimensional vectors. To resolve this problem, Lodhi et al. (2002) describe a direct algorithm for computing the kernel (inner product between string representations in feature space) in $O(n|s||t|)$ time, where n is subsequence length, and $|s|$ and $|t|$ are string lengths. However, since in our work we focus on strings that are much shorter than typical text documents, storing an explicit vector-space representation is computationally feasible with appropriate hashing data structures.

  We plan to use subsequence feature spaces for obtaining pair vectors that would be classified by the support vector machine yielding similarity between strings. Since this representation yields significantly denser vectors than simple tokenization, we conjecture that it will enable us to avoid overly aggressive feature selection by the SVM, which will in turn allow using the bias reduction technique described below.

- There are two parameters in support vector machines with Gaussian kernels that account for bias. First, the regularization parameter C controls the tradeoff between obtaining the maximum possible margin of the separating hyperplane and fitting all training data to be separated. Second, kernel width σ^2 determines how much influence individual training points exert on the discriminant function learned by the classifier, implicitly controlling the number of support vectors.

  In recent work, Valentini and Dietterich (2002) described a technique for analyzing the bias-variance decomposition for support vector machines. In their experiments, they found that an optimal region in the space of SVM parameters exists where both variance and bias are low. Following this work, we will incorporate a simple parameter search procedure (e.g. using Powell's linear search) into the SVM training algorithm to obtain values of C and σ^2 that correspond to a low-bias SVM that will then yield an accurate token-based string similarity function.

- When string similarity functions are used for the task of identifying equivalent strings, we are only concerned with the correct ordering of string pairs by their similarity. Converting the SVM margins to similarity values using Equation (10) served this purpose; however, a more principled approach to obtaining string similarity values from the classification margin for the pair vector would be appropriate.

  One approach to calculating similarity values from the classification margin can rely on converting the SVM output to a true probabilistic value. Platt (1999) has suggested using a sigmoid function for such conversion, where the sigmoid parameters are fitted using maximum likelihood estimation. We plan to incorporate Platt's method into our approach, which would allow us to obtain a learnable vector-space string similarity measure with a clear probabilistic interpretation similar to one provided by generative models for string edit distance.

Since these three approaches tackle orthogonal problems with our current technique, we hope that employing a combination of them will make learnable vector-space similarity based on pairwise classification a useful adaptive method for string distance computation.

Learning Vector-space Similarity Using Gradient Descent Along with the pairwise classification approach to vector-space string similarity, we intend to look at other methods for adapting vector-space similarity computations to a particular domain using string labeled pairs. Cohn et al. (2000) employed gradient descent for weighting the contribution of individual words to Jensen-Shannon divergence for clustering documents, effectively training a probabilistic similarity function for clustering. We plan to experiment with gradient descent methods for learning feature weights, possibly utilizing methods such as simulated annealing, and compare this approach with classification-based vector-space similarity.

4.1.2 Extending String Edit Distance with Affine Gaps

While learnable string edit distance with affine gaps leads to substantial improvements over untrained string metrics, we plan to investigate several additional enhancements to string edit distance that we hope will yield a more accurate metric.

One such improvement involves detecting and learning weights for macros, which can be defined as frequent contiguous string subsequences that can be treated as atomic objects for the purpose of computing edit distance. For example, subsequence *"reet"* can be treated as a single unit for the purpose of comparing street addresses (corresponding to a frequent transformation *"Street"*→ *"St"*).

We propose to incorporate macros into string edit distance with affine gaps by automatically detecting them in a training corpus of equivalent strings and including them into the learning algorithm. Given a training set of equivalent string pairs $\mathcal{T} = \{(s_1, t_1), ...(s_n, t_n)\}$, macros can be identified by training learnable string edit distance with affine gaps as described in Section 3.2.1, computing optimal alignments between the equivalent strings in $\mathcal{T}$, and selecting the most frequent insertion/deletion sequences. Then, string edit distance with affine gaps can be re-trained treating macros as atomic symbols and learning edit operation weights for them. The optimum threshold frequency for macro detection can be selected by cross-validation on the training corpus.

Our approach for learning string edit distance with affine gaps relies on training using a corpus of equivalent string pairs. It is possible, however, that training data may include pairs of strings that are explicitly *not* equivalent. One way to incorporate negative training examples in the learning algorithm described in Section 3.2.1 is to use discriminative training. Bahl, Brown, de Souza, and Mercer (1986) described the Maximum Mutual Information (MMI) method for training hidden Markov models to discriminate between positive and negative examples in a maximum a posteriori (MAP) framework. We hope to utilize their approach for learning string edit distance using both positive and negative string pair examples.

4.1.3 CRF-based String Edit Distance

Conditional random fields, recently introduced by Lafferty et al. (2001), are an undirected graphical framework for probabilistic sequence labeling that yields significant improvements over other graphical models such as hidden Markov models (HMMs). Conditional random fields have been shown to outperform traditional approaches on such tasks as part-of-speech tagging (Lafferty et al., 2001), named entity recognition (McCallum & Li, 2003), table extraction (Pinto, McCallum, Wei, & Croft, 2003), and identity uncertainty (McCallum & Wellner, 2003). Key benefits of conditional random fields over other probabilistic models are the following.

- Conditional random fields attempt to assign most likely labels to observation sequences without modeling the generative process that produced the observations. More specifically, instead of estimating the joint probability of paired observation and label sequences, conditional random fields only compute the probabilities of possible label sequences. As a result, no computational effort is spent on modeling the observations, which allows performing inference using multiple non-independent features.

- Unlike other conditional models such as maximum entropy Markov models (McCallum, Freitag, & Pereira, 2000) and maximum entropy taggers (Ratnaparkhi, 1999), conditional random fields do not suffer from the *label bias* problem. The label bias problem occurs because per-state normalization of transition probabilities leads to all probability mass arriving at a state being passed along the transitions, while observations are effectively ignored. Conditional random fields avoid this problem by using a single exponential model for the joint probability of the entire sequence of labels. This trades off the weights of different features at different states against each other, leading to increased accuracy on tasks such as part-of-speech tagging (Lafferty et al., 2001).

- Observations in HMMs are atomic entries, such as character pairs or words. Representing multiple interacting features or long-range dependencies is not practical since it makes solving the inference problem for the models intractable. In contrast, conditional random fields allow constructing specialized, long-range features that can be highly informative for the prediction task.

In future work, we hope to employ conditional random fields for modeling edit distance between strings. We are currently investigating mechanisms for formalizing string alignment as a random field, which will allow us to construct probabilistic models of string edit distance that are much more expressive than the HMM-based models of Ristad and Yianilos (1998) and Section 3.2.1.

4.1.4 Learning Semantic String Similarity Using External Sources

In many domains, strings that are closely related or identical may have very different syntactic representations. For example, geographic locations *"Hollywood"* and *"Los Angeles"* can be synonymous for address matching, while string distance measures described above view them as highly dissimilar. We propose to use *string context* for estimating semantic similarity: since strings that consistently appear in similar contexts are likely to be related, measuring similarity between the contexts provides us with valuable information about string likeness.

Sometimes an extensive domain model that allows conducting type-specific string comparisons is available (Michalowski, Thakkar, & Knoblock, 2003). With geographical addresses, for example, mapping street addresses and zip codes to latitude and longitude provides physical proximity information. However, domain-specific secondary sources may not be available. In such situations, semantic similarity of strings can be inferred from large external sources such as large text corpora or the World Wide Web. We propose the following three measures for obtaining semantic similarity from external sources:

- **Average context cosine similarity**. Given a set of documents $\mathcal{S}_s$ in which the string s occurs, a prototype vector $\mathbf{c_s}$ for the context of s can be constructed by averaging the TF-IDF vector-space representations for each document $\mathbf{d_s} \in \mathcal{S}_s$ in which s occurs (excluding s itself from the vector-space representation). Such prototypes are similar to those used in the Rocchio text classifier (Rocchio, 1971):

$$\mathbf{c_s} = \sum_{\mathbf{d_s} \in \mathcal{S}_s} \mathbf{d_s}$$

  Contextual similarity between strings s and t can then be approximated by cosine similarity between their prototype vectors:

$$Sim_{avg-context}(s,t) = \frac{\mathbf{c_s}^T \mathbf{c_t}}{\|\mathbf{c_s}\| \|\mathbf{c_t}\|} \quad (16)$$

- **Latent Semantic similarity.** Latent Semantic Analysis (LSA) is a well-known dimensionality reduction technique that uses singular value decomposition (SVD) of the term-document matrix to find a projection of the original vector-space into a lower-dimensional space that is optimal in the least-squared error sense. Performing SVD on the term-document matrix $\mathbf{X}$ encoding d documents in vector-space of dimensionality m results in a decomposition $\mathbf{X} = \mathbf{T}_{m \times n} \mathbf{S}_{n \times n} \mathbf{D}^T_{d \times n}$, where $n = \min(m, d)$. Matrix $\mathbf{T}$ contains projections of original terms into the new representation, and similarity can be measured as the cosine of the angle between the projections of terms into the new space (Landauer & Dumais, 1997).

- **Jensen-Shannon similarity.** Assuming that documents are generated by a multinomial model that follows the naive Bayes assumption for words/features, we can develop a probabilistic measure of string similarity. In the multinomial model, we assume that every document is an ordered sequence of independent word events (Lewis & Catlett, 1994; McCallum & Nigam, 1998). Then, for each string

s, probability of encountering token w_i in the same document can be estimated from co-occurrence counts:

$$p(w_i|s) = \frac{1 + \sum_{d_s \in \mathcal{S}_s} N(w_i, d_s)}{|\mathcal{V}| + \sum_{j=1..|\mathcal{V}|} \sum_{d_s \in \mathcal{S}_s} N(w_j, d_s)}$$

where $\mathcal{S}_s$ is the set of documents where string s occurs, and $N(w_i, d_s)$ is the number of occurrences of word w_i in document d_s. Thus, each string s can be described by the conditional probability distribution $p_s(w) = p(w|s)$ over the vocabulary $\mathcal{V}$. Then, for any two strings s and t, Jensen-Shannon divergence can be used as a measure of similarity between the corresponding probability distributions $p_s(w)$ and $p_t(w)$ (Dhillon, Mallela, & Kumar, 2002):

$$Sim_{JS}(s,t) = \pi_s KL(p_s, \pi_s p_s + \pi_t p_t) + \pi_t KL(p_t, \pi_s p_s + \pi_t p_t) \tag{17}$$

where π_s and π_t are relative prior probabilities of s and t estimated from occurrence counts of s and t in the corpus: $\pi_s = \frac{N(s)}{N(s)+N(t)}$ and $\pi_t = \frac{N(t)}{N(t)+N(s)}$; and $KL(p_s, \pi_s p_s + \pi_t p_t)$ is Kullback-Leibler divergence to the mean probability distribution: $KL(p_s, \pi_s p_s + \pi_t p_t) = \sum_{w_i \in V} p_s(w_i) \log \frac{p_s(w_i)}{p(w_i|\pi_s s + \pi_t t)}$. Thus, we suggest estimating semantic similarity of strings s and t based on external sources as likeness between the probabilistic models for terms occurring in their context.

We foresee two possible scenarios for computing semantic string similarity from external sources. If a large corpus of text documents or a database of text records is available, similarity functions described above can be computed using such a corpus or a database. It is possible, however, that certain strings occur infrequently in the provided dataset. For this scenario, we propose using the World Wide Web as a large text corpus that is accessible via a search engine. When similarity between strings s and t needs to be estimated, each of the strings can be supplied as a query to a search engine. Documents returned as search results will then form sets $\mathcal{S}$ and $\mathcal{T}$ based on which contextual similarity can be estimated using the three measures described above.

In previous work, Turney (2001) used statistical data obtained by querying a Web search engine for recognizing synonyms, while Bunescu (2003) proposed a similar approach for anaphora resolution in natural text. However, their methods only considered similarity as a function of the number of common documents returned by the search engine. We propose to go a step further and use the actual contents of documents retrieved from the Web, which will allow us to use context-specific similarity measures described above. We conjecture that such measures will perform significantly better than statistics that only compare retrieval results.

Since semantic similarity based on external sources can be expensive to compute compared to standard syntactic distance metrics, we propose to use it in record linkage scenarios only as an additional similarity feature in conjunction with standard syntactic measures. Combining multiple similarity measures can be performed as a part of the two-layer learning framework described in Section 3.3.1, where record similarity function is trained after learning individual field similarity functions. We hope to show that using external information sources can improve string comparisons for the record linkage task and lead to more accurate identification of equivalent records.

4.2 Clustering with Learnable Vector-space Similarity Functions

Our initial experiments with learning similarity functions for semi-supervised clustering described in Section 3.4 show that clustering algorithms can benefit from utilizing metrics learned from pairwise training data.

While we proposed a method for combining learnable Euclidean distance with seeding, in future work we plan to investigate learning methods for weighted cosine similarity. This will allow us to work with high-dimensional data that is common in real-world domains such as text and computational biology.

The vector-space learnable similarity based on pairwise classification described in Section 3.2.2 can be employed for any domain where data is represented by vectors in Euclidean space. Along with approaches proposed above for improving this similarity function, we will evaluate its applicability to semi-supervised clustering using algorithms such as hierarchical agglomerative clustering (HAC) and k-means, as well as clustering algorithms based on graph partitioning techniques (Karypis & Kumar, 1998; Meila & Shi, 2001). We will employ text clustering datasets such as *20Newsgroups*[3] and *DMOZ* (Dhillon et al., 2002) to evaluate our methods for high-dimensional similarity function learning.

4.3 Active learning techniques for similarity metrics

As we have discussed in Section 3.3.4, active learning for learnable similarity functions aims to identify pairs of objects whose labels are most informative for improving distance computations. For similarity metrics that are used in clustering, active learning methods seek pairs of objects that the user labels as belonging to the same cluster or different clusters.

The classifier-based learnable vector-space similarity described in Section 3.2.2 lends itself nicely to active learning techniques developed for classification. Prior work on active learning for support vector machines has shown that SVM accuracy can increase significantly when training examples are sampled selectively. Tong (2001) proposed a method that selects examples which lead to maximum reduction in version space, while Schohn and Cohn (2000) developed a technique similar to uncertainty sampling that selects examples which are closest to the separating hyperplane of the SVM. Finally, query-by-committee active learning has been successfully applied to SVMs for record linkage by Sarawagi and Bhamidipaty (2002). We plan to investigate the utility of these methods for reducing the number of training examples required to learn a vector-space classifier-based distance metric accurately.

As our initial investigation of training example selection strategies for record linkage has shown, the biggest challenge in selecting useful training example pairs lies with the fact that the space of possible pairs grows quadratically with the number of examples, while most pairs are non-equivalent and not useful for training metrics. In future work, we intend to continue developing static-active strategies for string metrics by incorporating more principled uncertainty sampling techniques (Lewis & Catlett, 1994), as well as comparing static-active and dynamic-active methods. We also plan to combine static-active techniques with semi-supervised strategies that use weakly labeled examples. Additionally, we will experiment with purely unsupervised discriminative training of similarity functions using weakly-labeled negative examples along with weakly-labeled positive examples. We will compare this approach with previously developed unsupervised record linkage techniques that use the Expectation-Maximization algorithm (Winkler, 1993).

In previous work, Dagan and Engelson (1995) have proposed a query-by-committee active learning method for hidden Markov models using committees of models created by varying HMM parameters. We are interested in utilizing their method for actively selecting string pairs for training string edit distance with affine gaps. We hope that combining active learning techniques with discriminative training of HMMs (Bahl et al., 1986) can drastically lower the number of examples required to learn reliable parameters for string edit distances and thereby yield accurate metrics.

[3] http://www.ai.mit.edu/people/jrennie/20Newsgroups

5 Conclusion

Accurate similarity functions are important for many learning problems including clustering and duplicate detection. Developing functions that can adapt to a particular domain is a promising direction for research. Our initial results on the record linkage task with learnable string similarity functions show improvements over their static analogs, and we hope to obtain even better performance with several enhancements to our methods. We also propose new adaptive similarity measures based on conditional random fields and information contained in large text corpora. We demonstrate good performance of our proposed approach for incorporating learnable similarity functions into semi-supervised k-means clustering with seeding. We plan to continue this line of research by focusing on learnable similarity functions for high-dimensional vector data. Finally, we intend to extend our work on active learning methods for selecting informative training examples for learning similarity metrics.

References

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. ACM Press, New York.

Bahl, L., Brown, P. F., de Souza, P. V., & Mercer, K. L. (1986). Maximum mutual information estimation of hidden markov parameters for speech recognition. In *Proceedings of IEEE ICASSP '86*, pp. 49–52.

Basu, S., Banerjee, A., & Mooney, R. J. (2002). Semi-supervised clustering by seeding. In *Proceedings of 19th International Conference on Machine Learning (ICML-2002)*, pp. 19–26.

Basu, S., Banerjee, A., & Mooney, R. J. (2003a). Active semi-supervision for pairwise constrained clustering. Submitted for publication, available at `http://www.cs.utexas.edu/~sugato/`.

Basu, S., Bilenko, M., & Mooney, R. J. (2003b). Comparing and unifying search-based and similarity-based approaches to semi-supervised clustering. In *Proceedings of the ICML-2003 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, Washington, DC.

Baxter, J. (1997). The canonical distortion measure for vector quantization and function approximation. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97)*, pp. 39–47, San Francisco, CA.

Bilenko, M., & Mooney, R. J. (2002). Learning to combine trained distance metrics for duplicate detection in databases. Tech. rep. AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX.

Bilenko, M., & Mooney, R. J. (2003a). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pp. 39–48, Washington, DC.

Bilenko, M., & Mooney, R. J. (2003b). On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, Washington, DC.

Bilmes, J. (1997). A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Tech. rep. ICSI-TR-97-021, ICSI.

Boykov, Y., Veksler, O., & Zabih, R. (1998). Markov random fields with efficient approximations. In *IEEE Computer Vision and Pattern Recognition Conf.*

Bunescu, R. (2003). Associative anaphora resolution: A web-based approach. In *Proceedings of the EACL-2003 Workshop on the Computational Treatment of Anaphora*, pp. 47–52, Budapest, Hungary.

Cohen, W., & Richman, J. (2001). Learning to match and cluster entity names. In *ACM SIGIR-2001 Workshop on Mathematical/Formal Methods in Information Retrieval*, New Orleans, LA.

Cohen, W. W. (1998). Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, pp. 201–212, Seattle, WA.

Cohen, W. W., Kautz, H., & McAllester, D. (2000). Hardening soft information sources. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, Boston, MA.

Cohen, W. W., Ravikumar, P., & 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*, pp. 73–78, Acapulco, Mexico.

Cohen, W. W., & 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)*, Edmonton, Alberta.

Cohn, D., Caruana, R., & McCallum, A. (2000). Semi-supervised clustering with user feedback. Unpublished manuscript. Available at `http://www.cs.umass.edu/~mccallum/`.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, *4*, 129–145.

Comon, P. (1994). *Independent component analysis*, a new concept?. *Signal Processing*, *36*, 287–314.

Dagan, I., & Engelson, S. P. (1995). Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pp. 150–157, San Francisco, CA. Morgan Kaufmann.

Demiriz, A., Bennett, K. P., & Embrechts, M. J. (1999). Semi-supervised clustering using genetic algorithms. In *ANNIE'99 (Artificial Neural Networks in Engineering)*.

Dhillon, I. S., & Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. *Machine Learning*, *42*, 143–175.

Dhillon, I., Mallela, S., & Kumar, R. (2002). Enhanced word clustering for hierarchical classification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton.

Domeniconi, C., & Gunopulos, D. (2002). Adaptive nearest neighbor classification using support vector machines. In *Advances in Neural Information Processing Systems 13*.

Domeniconi, C., Peng, J., & Gunopulos, D. (2001). An adaptive metric machine for pattern classification.. In *Advances in Neural Information Processing Systems 13*, pp. 458–464.

Domingos, P. (2000). A unified bias-variance decomposition and its applications. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 231–238, San Francisco, CA.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification* (Second edition). Wiley, New York.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Friedman, J. (1994). Flexible metric nearest neighbor classification. Tech. rep. 113, Stanford University Statistics Department.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, *1*(1), 55–77.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, *162*, 705–708.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, New York.

Hastie, T., & Tibshirani, R. (1996). Discriminant adaptive nearest-neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *18*(6), 607–617.

Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD-95)*, pp. 127–138, San Jose, CA.

Hofmann, T., & Buhmann, J. M. (1998). Active data clustering. In *Advances in Neural Information Processing Systems 10*, pp. 528–534.

Hyvärinen, A., & Oja, E. (2000). Independent component analysis: Algorithms and applications. *Neural Networks*, *13(4-5)*, 411–430.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review.. *ACM Computing Surveys*, *31*(3), 264–323.

Jaro, M. (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.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C. J. C., & Smola, A. J. (Eds.), *Advances in Kernel Methods - Support Vector Learning*, pp. 169–184. MIT Press.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer-Verlag, New York.

Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, *20*(1), 359–392.

Kaufman, L., & Rousseeuw, P. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, New York.

Kleinberg, J., & Tardos, E. (1999). Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. In *IEEE Symp. on Foundations of Comp. Sci.*

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of 18th International Conference on Machine Learning (ICML-2001)*.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, *104*, 211–240.

Lawrence, S., Bollacker, K., & Giles, C. L. (1999). Autonomous citation matching. In *Proceedings of the Third International Conference on Autonomous Agents*, New York, NY. ACM Press.

Lee, D. D., & Seung, H. S. (1997). Unsupervised learning by convex and conic coding. In *Proceedings of the Conference on Neural Information Processing Systems*, pp. 515–521.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*, 788–791.

Levenshtein, V. I. (1966). Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Lewis, D. D., & Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML-94)*, pp. 148–156, San Francisco, CA. Morgan Kaufmann.

Lindenbaum, M., Markovitch, S., & Rusakov, D. (1999). Selective sampling for nearest neighbor classifiers.. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 366–371.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, *2*, 419–444.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, Stanford, CA.

McCallum, A., & Li, W. (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proc. of the Seventh Conference on Natural Language Learning (CoNLL-2003)*, Edmonton, Canada.

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *Papers from the AAAI-98 Workshop on Text Categorization*, pp. 41–48, Madison, WI.

McCallum, A., Nigam, K., & Ungar, L. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pp. 169–178, Boston, MA.

McCallum, A., & Wellner, B. (2003). Toward conditional models of identity uncertainty with application to proper noun coreference. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pp. 79–86, Acapulco, Mexico.

Meila, M., & Shi, J. (2001). Learning segmentation by random walks. In *Advances in Neural Information Processing Systems 13*, pp. 873–879.

Michalowski, M., Thakkar, S., & Knoblock, C. A. (2003). Exploiting secondary sources for automatic object consolidation. In *Proceedings of the ACM SIGKDD-03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pp. 34–36, Washington, DC.

Monge, A. E., & Elkan, C. (1996). The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pp. 267–270, Portland, OR.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 23–29, Tuscon, AZ.

Muslea, I. (2002). *Active learning with multiple views*. Ph.D. thesis, University of Southern California.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, *48*, 443–453.

Newcombe, H. B., Kennedy, J. M., Axford, S. J., & James, A. P. (1959). Automatic linkage of vital records. *Science*, *130*, 954–959.

Phillips, P. J. (1999). Support vector machines applied to face recognition. In Kearns, M. J., Solla, S. A., & Cohn, D. A. (Eds.), *Advances in Neural Information Processing Systems 11*. MIT Press.

Pinto, D., McCallum, A., Wei, X., & Croft, W. B. (2003). Table extraction using conditional random fields. In *Proceedings of 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Toronto, Canada.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Smola, A. J., Bartlett, P., Schölkopf, B., & Schuurmans, D. (Eds.), *Advances in Large Margin Classifiers*, pp. 185–208. MIT Press.

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

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, *34*, 151–176.

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

Rocchio, J. (1971). Relevance feedback in information retrieval. In Salton, G. (Ed.), *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 313–323. Prentice Hall.

Roy, N., & McCallum, A. (2001). Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of 18th International Conference on Machine Learning (ICML-2001)*.

Salton, G., & McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw Hill, New York.

Sarawagi, S., & Bhamidipaty, A. (2002). Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta.

Schohn, G., & Cohn, D. (2000). Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 839–846. Morgan Kaufmann, San Francisco, CA.

Seung, H. S., Opper, M., & Sompolinsky, H. (1992). Query by committee. In *Proceedings of the ACM Workshop on Computational Learning Theory*, Pittsburgh, PA.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*, 195–197.

Strehl, A. (2002). *Relationship-based clustering and cluster ensembles for high-dimensional data mining*. Ph.D. thesis, The University of Texas at Austin.

Strehl, A., Ghosh, J., & Mooney, R. (2000). Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pp. 58–64.

Tejada, S., Knoblock, C. A., & Minton, S. (2001). Learning object identification rules for information integration. *Information Systems Journal*, *26*(8), 635–656.

Tejada, S., Knoblock, C. A., & Minton, S. (2002). Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, Edmonton, Alberta.

Tong, S. (2001). *Active Learning: Theory and Applications*. Ph.D. thesis, Stanford University, Stanford, CA.

Turney, P. D. (2001). Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning (ECML-01)*, pp. 491–502, Freiburg, Germany.

Tversky, A. (1977). Features of similarity. *Psychological Review*, *84*(4), 327–352.

Valentini, G., & Dietterich, T. G. (2002). Bias-variance analysis and ensembles of svm. In *Third International Workshop on Multiple Classifier Systems (MCS-2002)*, pp. 222–231, Cagliari, Italy.

Valentini, G., & Dietterich, T. G. (2003). Low bias bagged support vector machines. In *Proceedings of 20th International Conference on Machine Learning (ICML-2003)*, pp. 752–759, Washington, DC.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley & Sons.

Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S. (2001). Constrained K-Means clustering with background knowledge. In *Proceedings of 18th International Conference on Machine Learning (ICML-2001)*.

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pp. 354–359.

Winkler, W. E. (1993). Improved decision rules in the fellegi-sunter model of record linkage. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Winkler, W. E. (1994). Advanced methods for record linkage. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Winkler, W. E. (1999). The state of record linkage and current research problems. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2003). Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*. MIT Press.