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	**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: $cost(s[i{:}j]) = \sigma c(s[i], \epsilon) + \delta \sum_{k=i+1\ldots j} c(s[k], \epsilon)$, 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:

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

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

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

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

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:

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

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:

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

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

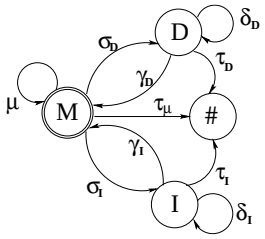

Figure 2: A generative model for edit distance with affine gaps

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