Active Learning of Domain-Specific Distances for Link Discovery

Tommaso Soru and Axel-Cyrille Ngonga Ngomo

Department of Computer Science
University of Leipzig
Augustusplatz 10, 04109 Leipzig
{tsoru,ngonga}@informatik.uni-leipzig.de
http://limes.sf.net

Abstract. Discovering cross-knowledge-base links is of central importance for manifold tasks across the Linked Data Web. So far, learning link specifications has been addressed by approaches that rely on standard similarity and distance measures such as the Levenshtein distance for strings and the Euclidean distance for numeric values. While these approaches have been shown to perform well, the use of standard similarity measure still hampers their accuracy, as several link discovery tasks can only be solved sub-optimally when relying on standard measures. In this paper, we address this drawback by presenting a novel approach to learning string similarity measures concurrently across multiple dimensions directly from labeled data. Our approach is based on learning linear classifiers which rely on learned edit distance within an active learning setting. By using this combination of paradigms, we can ensure that we reduce the labeling burden on the experts at hand while achieving superior results on datasets for which edit distances are useful. We evaluate our approach on three different real datasets and show that our approach can improve the accuracy of classifiers. We also discuss how our approach can be extended to other similarity and distance measures as well as different classifiers.

1 Introduction

Discovering cross-knowledge-base links is of central importance to realize the vision of the Linked Data Web [1]. Over the last years, several frameworks and approaches have been developed to address the two main hurdles of link discovery: the quadratic runtime [16,10] and the discovery of accurate link specifications [18,19]. In both cases, most approaches assume that link discovery can be carried out by devising a similarity function σ such that instances from a source and target knowledge base whose similarity is larger than a certain threshold θ should be linked. In most cases, σ is a combination of several atomic similarity measures, i.e., of measures that compare solely one pair of property values from the source and target instances. The combination (σ, θ) is usually called link specification [16] or linkage rule [10]. So far, the detection of accurate link specification was carried out by using techniques such as linear or Boolean classifier

H. Takeda et al. (Eds.): JIST 2012, LNCS 7774, pp. 97–112, 2013.

Table 1. Example of property values

Source labels	Target labels
Adenomatoid tumor	Adenomatoid tumour
Odontogenic tumor	Odontogenic tumour
Generalised epidermolysis	Generalized epidermolysis
Diabetes I	Diabetes I
Diabetes II	Diabetes II
Anemia Type I	Anemia Type I
Anemia Type II	Anemia Type II
Anemia Type III	Anemia Type III

learning [17] or genetic programming [11,18,19]. However, only generic string similarity measures have been used as atomic measures within these learning approaches. In this paper, we go beyond the state of the art by not only learning classifiers but also learning the similarity metrics used in these classifiers directly from the data. In addition, we go beyond the state-of-the-art in metric learning by applying an active learning approach [24].

The need for our approach is motivated by the exemplary link discovery task in the medical domain showed in Table 1. The source and target contain property values are written in American and British English and thus display minor differences (e.g., "Generalized" vs. "Generalised"). Therewith intertwined are yet also semantic differences (e.g., "Diabetes I" vs. "Diabetes II") which correspond to exactly the same edit distance between the property values. Consequently, if a user was to compute links between these two data sets by means of these property values using the edit distance, he/she could either (1) choose to set the distance threshold to 0, which would lead to a precision P = 1 but to a recall of R = 0.625 (F = 0.77) or (2) choose to set the threshold to 1, which would lead to R = 1 but to P = 0.57 (F = 0.73). It is obvious that for this particular example, there is no distance threshold for the standard edit distance that would lead to an F-measure of 1. The same holds for other generic string similarity measures such as Jaccard. Consequently, none of the previous approaches (see e.g. [19,18]) to learning link specifications would be able to detect an optimal link specification for the data at hand. The basic intuition behind this work is the following: If we could model that replacing "s" by"z" yields less semantics that inserting the token "I" for this particular pair of property values, we could assign different weights to these tokens during the computation of the edit distance. It is important to note that this difference would only be valid for this particular pair of properties. For standardized labels such as company names, the substitution of "s" by"z" might yield exactly the same semantics as inserting an "I". Given that most link specifications rely on a complex similarity functions (i.e., on combination of several similarity values) to compute links (e.g., similarity of label, longitude and latitude for cities, etc.), the additional requirement of devising *a domain-specific similarity measure for each of the property pairs used* for linking concurrently is of central importance. Hence, the main goal of this work is to devise a supervised approach for learning link specifications which rely

on complex domain-specific similarities learned directly from the input data. As learning such a metric can require a considerable amount of human labeling, we implement our metric learning approach within the pool-based setting of active learning [24] to minimize the burden on human annotators. In the rest of this paper, we focus on learning similarities based on generalized edit distances. In the discussion section, we show how our approach can be extended to learning other measures such as weighted Jaccard and n-gram-based similarities. Our main contributions are thus as follows:

- We present a supervised machine for learning link specifications based on domain-specific similarities.
- To reduce the burden on the user, we learn these similarities by using active learning.
- We show how our framework can be extended to other similarities such as Q-Grams and Jaccard.
- We evaluate the learned similarities against three different data sets and analyze the strengths and weaknesses of our approach [1].

2 Preliminaries and Notation

2.1 Link Discovery as Classification

In this work, we regard link discovery as a classification problem. Let S (source) and T (target) be two sets of resources. Furthermore and without loss of generality, let each resource from S and T be described by a tuple of n property values such that the i^{th} property value of elements of S maps the i^{th} property value of elements of T.[2] The general aim of link discovery is to devise an (mostly with respect to its F-score) accurate classifier $C : S \times T \rightarrow \{-1, +1\}$ that assigns the class $+1$ to a pair (s, t) should be linked and -1 else. Most approaches for link discovery rely on a combination of operators (such as minimum, maximum and linear combinations), similarity measures σ_i (such as Jaccard and Cosine) that compare property values and similarity thresholds θ_i to determine such as classifier. In previous works, only the operators and thresholds were learned [17,11,18,19]. Yet, as shown by our example, such approaches are not always sufficient to achieve a high F-measure. We go beyond the state of the art by learning the similarity measures σ_i. In this work, we focus on learning these measures in combination with weighted linear classifiers, i.e., classifiers such that

$$C(s,t) = +1 \Leftrightarrow \left(\sum_{i=1}^{n} w_i \sigma_i(s,t) - \theta \geq 0 \right), \tag{1}$$

where σ_i is a similarity function that compares the values of the i^{th} property of s and t and aim to learn the values of w_i, θ and σ_i concurrently. Note that other

[1] The implementation of our approach is publicly available at `http://metric-learning.googlecode.com`

[2] Note that such pairs can be calculated using schema matching approaches such as those described in [20].

classifiers can be combined with our approach. For a given classifier C, we call (s, t) *positive* if $C(s, t) = +1$ and *negative* else. Also note that while we focus on string similarities in this work, our implementation also supports numerical values by the means of learning Mahalanobis distances.

Learning classifiers within the context of active learning means relying on user feedback after each iteration to improve the classifiers until a termination condition is met. Given the type of classifier we use, we will assume the pool-based sampling setting [24]. We will denote learning iterations with the variable τ. We will denote classifiers, weights, thresholds, matrices and similarities at iteration τ by using superscripts. Consequently, we will label the input generated by the user at iteration τ with Q^τ. The classifier and distance matrix computed at the same iteration will be denoted C^τ resp. M^τ. Note that in the following, we limit ourselves to learning generalized edit similarities σ_i, which we define as $(1 + \delta_i)^{-1}$, where δ_i is a generalized edit distance. We chose to learn similarities instead of distances because they are bound by finite values, making the computation of the classifier more efficient. However, the basic idea behind our approach can be extended to other similarities and distances with ease. We thus present generalization of other types of similarity measures and show how our approach can be extended to them in Section 6.

2.2 Weighted Edit Distance

Let A and B be two strings over an alphabet Σ. Furthermore, let $\Sigma^* = \Sigma \cup \{\epsilon\}$, where ϵ stand for the empty string. The edit distance $\delta(A, B)$ (also called Levenshtein distance [13]) is defined as the minimal number of edit operations (insertion, deletion and substitution) necessary to transform the string A into B. Thus, e.g., $\delta($"Diabetes I", "Diabetes II"$) = \delta($"Adenomatoid tumor", "Adenomatoid tumour"$) = 1$. Previous work from the machine learning community (see e.g., [3]) has proposed learning the cost of each possible insertion, deletion and substitution. Achieving this goal is equivalent to learning a positive cost matrix M of size $|\Sigma^*| \times |\Sigma^*|$. The rows and columns of M are issued from Σ^*. The entry m_{ij} stands for the cost of replacing the i^{th} character of Σ^* with the j^{th} character of Σ^*. Note that the deletion of a character $c \in \Sigma$ is modeled as replacing c with ϵ, while insertion of c is modeled as replacing ϵ with c. Within this framework, the standard edit distance can be modeled by means of a cost matrix M such that $m_{ij} = 1$ if $i \neq j$ and 0 else. The distance which results from a non-uniform cost matrix[3] is called a weighted (or generalized) edit distance δ_w. The distance $\delta_w(A, B)$ can now be defined as the least expensive sequence of operations that transforms A into B.[4] Note that we also go beyond the state of the art in metric learning by combining metric learning and active learning,

[3] For distances, a non-uniform matrix M is one such that $\exists i, j, i', j' : (i \neq j \wedge i' \neq j' \wedge m_{ij} \neq m_{i'j'})$.

[4] Note that while the edit distance has been shown to be a metric, M being asymmetric leads to δ_w not being a metric in the mathematical sense of the term, as $\exists A, B : \delta_w(A, B) \neq \delta_w(B, A)$.

which has been shown to be able to reduce the amount of human labeling needed for learning link specifications significantly [18].

3 The ACIDS Approach

The aim of our approach, ACIDS (Active Learning of Distances and Similarities), is to compute domain-specific similarity measures while minimizing the annotation burden on the user through active learning. Note that while we focus on learning similarity functions derived from the generalized edit distance, we also show how our approach can be generalized to other similarity measures.

3.1 Overview

Our approach consists of the following five steps:

1. $\tau = 0$; get Q^0 and compute $C^0, M^0_{1...n}$;
2. If all elements of $\bigcup_{i=0}^{\tau} Q^\tau$ can be separated by C^τ then goto step 4.
3. Else update the similarity measures σ_i^τ and update the classifier C^τ. If the maximal number of iterations is reached, goto step 4. Else goto step 2.
4. Increment τ. Select the k most informative positive and k most informative negative examples from $S \times T$ and merge these two sets to Q^τ.
5. Require labeling for Q^τ from the oracle. If termination condition reached, then terminate. Else goto step 2.

In the following, we elaborate on each of these steps.

3.2 Initialization

Several parameters need to be set to initialize ACIDS. The algorithm is initialized by requiring k positive and k negative examples $(s,t) \in S \times T$ as input. The positive examples are stored in P^0 while the negative examples constitute the elements of N^0. Q^0 is set to $P^0 \cup N^0$. For all n dimensions of the problem where the property values are strings, $\sigma_i^0(s,t) = (1 + \delta(s,t))^{-1}$ (where δ stands for the edit distance with uniform costs) is used as initial similarity function.

The final initialization step consists of setting the initial classifier C^0. While several methods can be used to define an initial classifier, we focus on linear classifiers. Here, the classifier is a hyperplane in the n-dimensional similarity space $\mathcal{S}$ whose elements are pairs $(s,t) \in S \times T$ with the coordinates $(\sigma_1^0(s,t), ..., \sigma_n^0(s,t))$. We set C^0 to

$$\sum_{i=1}^{n} \sigma_i^0(s,t) \geq n\theta_0. \tag{2}$$

This classifier is the hyperplane located at the Euclidean distance $\theta_0\sqrt{n}$ from the origin of $\mathcal{S}$ and at $(1-\theta_0)\sqrt{n}$ from the point of $\mathcal{S}$ such that all its coordinates are 1. Note that $\theta_0 \in [0,1]$ is a parameter that can be set by the user.

3.3 Computing Separability

The aim of this step is to check whether there is a classifier that can separate the examples we know to be positive from those we know to be negative. At each iteration τ, this step begins by computing the similarities $\sigma_i^\tau(s,t)$ of all labeled examples $(s,t) \in Q^\tau$ w.r.t. the current distance matrix M^τ. Note that $\sigma_i^\tau(s,t)$ varies with time as the matrix M^τ is updated. Checking the separability is carried out by using linear SVMs.[5] The basic idea behind this approach is as follows: Given a set of n-dimensional similarity vectors $x_1, ..., x_m$ and the classification $y_i \in \{+1, -1\}$ for each x_i, we define a classifier by the pair (w, b), where w is a n-dimensional vector and b is a scalar. The vector x is mapped to the class y such that

$$y(w^T x - \theta) \geq 0. \tag{3}$$

The best classifier for a given dataset is the classifier that minimizes $\frac{1}{2}w^T w$ subject to $\forall i \in \{1, ..., m\},\ y_i(w^T x_i - \theta) \geq 0$ [6]. In our case, the coordinates x_i of a point (s,t) at iteration τ are given by $\sigma_i^\tau(s,t)$. The components w_i of the vector w are the weights of the classifier C^τ, while θ is the threshold. If we are able to find a classifier with maximal accuracy using the algorithm presented in [5] applied to linear kernels, then we know the data to be linearly separable. In many cases (as for our toy data set for example), no such classifier can be found when using the original edit similarity function. In this case, the basic intuition behind our approach is that the similarity measures (and therewith the distribution of positive and negative examples in the similarity space) need to be updated in such a way that the positive examples wander towards the class $+1$ of the classifier, while the negative wander towards -1. Altering all M_i^τ in such a way is the goal of the subsequent step.

3.4 Updating the Similarity Measures

Overview. Formally, the basic idea behind our approach to updating similarity measures is that when given the two sets $P \subseteq S \times T$ of positive examples and $N \subseteq S \times T$ of negative examples, a good similarity measure σ_i^τ is one such that

$$\forall (s,t) \in P\ \forall (s',t') \in N\ \sigma_i^\tau(s,t) \geq \sigma_i^\tau(s',t'). \tag{4}$$

Let N^τ be the subset of Q^τ which contains all pairs that were labeled as false positives by the oracle at iteration τ. Furthermore, let P^τ be set of pairs labeled as false negatives during the same iteration. In addition, let $\mu_i^\tau : M \times S \times T \to [0,1]$ be a function such that $\mu_i^\tau(m_{ij}, s, t) = 1$ when the substitution operation modeled by m_{ij} was used during the computation of the similarity $\sigma_i^\tau(s,t)$ (the approach to computing μ is shown in the subsequent subsubsection). We update the weight matrix M_i^τ of σ_i^τ by employing a learning approach derived from perceptron learning. The corresponding update rule is given by

$$m_{ij}^\tau := m_{ij}^\tau - \sum_{(s,t)\in P^\tau} \eta^+ \mu_i^\tau(m_{ij}^\tau, s, t) + \sum_{(s',t')\in N^\tau} \eta^- \mu_i^\tau(m_{ij}^\tau, s', t'). \tag{5}$$

[5] Note that any other type of classifier could be used here.

While it might seem counter-intuitive that we augment the value of m_{ij} for negatives (i.e., points that belong to -1) and decrease it for positives, it is important to remember that the matrix M_i describes the distance between pairs of resources. Thus, by updating it in this way, we ensure that the operation encoded by m_{ij} becomes less costly, therewith making the pairs (s,t) that rely on this operation more similar. Negative examples on the other hand are a hint towards the given operation being more costly (and thus having a higher weight) than assumed so far. The similarity between two strings is then computed as $\sigma_i(s,t) = \frac{1}{1+\delta_i((s,t))}$, where the generic algorithm for computing weighted distances $\delta_i(A,B)$ for two strings A, B is shown in Algorithm 1. The algorithm basically computes the entries of the following matrix:

$$L_{ij}(A,B) = \begin{cases} i & \text{if } j = 0 \wedge 0 \leq i < |A| \\ j & \text{if } i = 0 \wedge 0 < j < |B| \\ \min\{L_{i-1,j}(A,B) + delCost(A[i]), & 0 < i < |A| \wedge 0 < j < |B| \\ L_{i,j-1}(A,B) + insCost(B[j]), & \\ L_{i-1,j-1}(A,B) + subCost(A[i],B[j])\} & \end{cases}, \quad (6)$$

where the computation of deletion, insertion and substitution costs for the same strings are as follows:

$$delCost(A[i]) := m_{pos(A[i]),pos(\epsilon)}, \quad (7)$$

$$insCost(A[j]) := m_{pos(\epsilon),pos(A[j])} \text{ and} \quad (8)$$

$$subCost(A[i],B[j]) := m_{pos(A[i]),pos(B[j])}. \quad (9)$$

Note that $A[i]$ is the i^{th} character of the string A, $B[j]$ is the j^{th} character of the string B and $pos(char)$ is the index of a character $char$ in the edit distance matrix.

Algorithm 1. Computation of Weighted Levenshtein distance

Require: $A, B \in \Sigma^*$
  $\forall i,j\ L[i,j] \leftarrow 0$
  for $i = 1 \rightarrow |A|$ **do**
    $L[i,1] \leftarrow i$
  end for
  for $j = 1 \rightarrow |B|$ **do**
    $L[1,j] \leftarrow j$
  end for
  for $i = 1 \rightarrow |A|$ **do**
    for $j = 1 \rightarrow |B|$ **do**
      $L[i,j] \leftarrow min\{L[i-1,j-1] + getSubCost(A[i],B[j]),$
        $L[i-1,j] + getDelCost(A[i]), L[i,j-1] + getInsCost(B[j])\}$
    end for
  end for
  return $L[|A|,|B|]$

Computation of μ. One of the key steps in our approach is the extension of the edit distance computation algorithm to deliver the sequence of operations that were used for determining the distance between (and therewith the similarity of) pairs (s, t). This computation has often been avoided (see e.g., [3]) as it is very time-consuming. Instead, the generic edit distance algorithm was used and the weights added to the path in a subsequent computation step. Yet, not carrying out this computation can lead to false weights being updated. Consider for example the computation of the edit distance between the two strings A = "BAG" and B = "BAGS" over a weighted matrix where $getInsCost(\text{"s"}) = getSubCost(\text{"s"}, \text{"S"}) = 0.3$. While the unweighted edit distance approach would carry out the insertion of "S" in A (cost = 1), it would actually be cheaper to carry out two operations: the insertion of "s" and the transformation of "s" to "S" (total costs: 0.6). To achieve the computation of such paths, we used the weighted distance computation algorithm shown in Algorithm 2. This algorithm basically runs the inverse of the dynamic programming approach to computing weighted edit distances and counts when which of the basic operations was used. It returns the sequence of operations that transformed A into B. Another approach to computing the shortest path would have been using Dijkstra's algorithm [7]. This alternative was evaluated by comparing the time complexity of both approaches. The combination of Algorithm 2 and 1 has a complexity of $O(mn) + O(m + n) \simeq O(mn)$ where $m = |A|$ and $n = |B|$. On the other hand, the best implementation of Dijkstra's algorithm, based on a min-priority queue using Fibonacci heap [8], runs in $O(|E| + |V| \log |V|)$. As the number of edges is mn and the number of vertices is $(m + 1)(n + 1)$, the time complexity is $O(mn + (m + 1)(n + 1) \log((m + 1)(n + 1))) \simeq O\left((m^2 + n^2) \log\left(m^2 + n^2\right)\right)$. Without loss of generality, the computational complexities can be reduced to $O\left(n^2\right)$ for our approach and $O\left(n^2 \log n\right)$ for Dijkstra's algorithm under the assumption that $n \geq m$. Consequently, we chose our less generic yet faster implementation of the computation approach for μ.

Computation of Learning Rates. Given the potential prevalence of negative examples (especially when one-to-one mappings are to be computed), we define η^- as $\max(|S|, |T|)\eta^+$. The intuition behind this definition is that in most cases, valid links only make up a small fraction of $S \times T$ as each s and each t are mostly linked to maximally one other resource. The probability for a random pair (s, t) of being a positive example for a link is then given by

$$P = \frac{\min(|S|, |T|)}{|S| \times |T|} = \frac{1}{\max(|S|, |T|)}. \tag{10}$$

Consequently the ratio of the positive and negative learning rates should be proportional to

$$\frac{\eta^+}{\eta^-} = \frac{P}{1 - P} \tag{11}$$

Given that P is usually very small, the equation reduces to

$$\eta^- \approx \max(|S|, |T|)\eta^+. \tag{12}$$

Algorithm 2. Computation of μ

Require: $L(A, B)$
  $L \leftarrow L(A, B)$
  $[i, j] \leftarrow \|L\| - 1$
  while $i \neq 0 \vee j \neq 0$ **do**
    if $j > 0$ **then**
      $left \leftarrow -L[i][j-1]$
    else
      $left \leftarrow \infty$
    end if
    if $i > 0$ **then**
      $up \leftarrow -L[i-1][j]$
    else
      $up \leftarrow \infty$
    end if
    if $i > 0 \wedge j > 0$ **then**
      $upleft \leftarrow -L[i-1][j-1]$
    else
      $upleft \leftarrow \infty$
    end if
    if $upleft \leq left \wedge upleft \leq up$ **then**
      $countSub(A[i-1], B[j-1])$ // substitution
      $i \leftarrow i - 1$
      $j \leftarrow j - 1$
    else
      if $left < upleft$ **then**
        $countIns(B[j-1])$ // insertion
        $j \leftarrow j - 1$
      else
        $countDel(A[i-1])$ // deletion
        $i \leftarrow i - 1$
      end if
    end if
  end while

The main implication of the different learning rates used by our approach is that the false negatives have a higher weight during the learning process as the probability of finding some is considerably smaller than that of finding false positives.

3.5 Determining the Most Informative Examples

Formally, the most informative examples are pairs $(s, t) \notin \bigcup_{i=0}^{\tau} Q^i$ that are such that knowing the right classification for these pairs would lead to the greatest improvement of the current classifier C^τ. As pointed out in previous work [17], these are the pairs whose classification is least certain. For linear classifiers, the

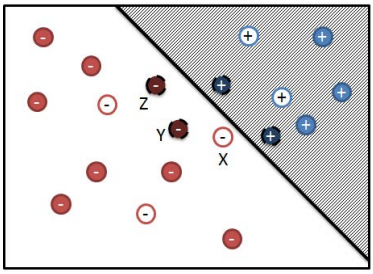


Fig. 1. Most informative examples

pairs (s,t) whose classification is least certain are obviously those elements from $S \times T$ with unknown classification that are closest to the boundary defined by C^τ. Figure 1 depicts this idea. The circles with a dashed border represent the 2 most informative positive and negative examples, the solid disks represent elements from $S \times T$ and the circles are examples that have already been classified by the user. Note that X is closer to the border than Y but is not a most informative example as it has already been classified.

Detecting the most informative examples would thus require computing the distance from all the pairs $(s,t) \in S \times T \setminus \bigcup_{i=0}^{\tau} Q^i$. Yet, this approach is obviously unpractical as it would require a computing almost $|S \times T|$ similarity values, given that $|\bigcup_{i=0}^{\tau} Q^i|$ is only a small fraction of $|S \times T|$. We thus devised the following approximation: Given the classifier defined by $\sum_{i=1}^{n} w_i \sigma_i(s,t) - \theta \geq 0$, we can derive the lower bound α_i for the minimal similarity $\sigma_i(s,t)$ that a pair (s,t) must achieve to belong to $+1$:

$$\sigma_i(s,t) \geq \alpha_i = \theta - \frac{\sum_{j=1, j \neq i}^{n} w_j \sigma_j(s,t)}{w_i}. \tag{13}$$

For each of the n dimensions of the similarity space, the most informative examples are thus likely to be found within the interval $[\alpha_i - \Delta, \alpha_i + \Delta]$, where $\Delta \in [0, 0.5]$. Consequently, we only need to compute the exact similarity scores of pairs (s,t) whose i^{th} coordinates lie within this interval. Several time-efficient approaches such as EdJoin [25] and PassJoin [14] have been developed to compute that set of pairs (s,t) such that $\delta(s,t)$ is smaller than a given threshold. Thus, if we never modified M_i, we could use these approaches to compute the pairs (s,t) with $\sigma_i(s,t) \in [\alpha_i - \Delta, \alpha_i + \Delta]$. Yet, given that we alter M_i and that the aforementioned approaches can only deal with uniform distance matrices M, we cannot only rely on them indirectly. Let $\kappa = \min_{m_{ij} \in M_i} m_{ij}$. Given $m \geq \kappa$ for all entries of m of M_i, the inequality $\delta_i(s,t) \geq \kappa \delta(s,t)$ holds. In addition, we know that by virtue of $\sigma_i = (1 + \delta_i)^{-1}$,

$$\sigma_i(s,t) \geq \alpha_i \Rightarrow \delta_i(s,t) \leq \frac{1-\alpha_i}{\alpha_i}. \quad (14)$$

Thus, we can approximate the set of pairs (s,t) with $\sigma_i(s,t) \geq \alpha_i - \Delta$ by

$$\sigma_i(s,t) \geq \alpha_i - \Delta \Rightarrow \delta(s,t) \leq \frac{1-(\alpha_i-\Delta)}{\kappa(\alpha_i-\Delta)}. \quad (15)$$

Consequently, we can run approaches such as EdJoin and PassJoin with the threshold $\frac{1-(\alpha_i-\Delta)}{\kappa(\alpha_i-\Delta)}$ when computing the set of pairs with $\sigma_i(s,t) \geq \alpha_i - \Delta$. Computing the set of pairs with $\sigma_i(s,t) \in [\alpha_i - \Delta, \alpha_i + \Delta]$ can then be achieved by simple filtering. Note that by using this approximation, we can discard a large number of pairs (s,t) for values of κ close to 1. Yet, this approximation is less useful when κ gets very small and is thus only of use in the first iterations of our approach. Devising an approach for the efficient computation of weighted edit distances remains future work.

3.6 Termination

Several termination conditions can be envisaged while running our approaches. Here, we chose to terminate the iteration when the classifier C^τ was able to classify all the entries of $Q^{\tau+1}$ correctly.

4 Experiments and Results

We began our experiments by running our approach on the toy example provided in Table 1. Our approach learned that the costs $getSubCost(\text{“s”}, \text{”z”}) = getInsCost(\text{“u”}) = 0.3$. Therewith, it was able to achieve an F-measure of 1.0. We then ran our approach on benchmark data as presented in the following.

4.1 Experimental Setup

We evaluated our approach on 3 real data sets (see Table 2)[6]. The first two of these data sets linked the publication datasets ACM, DBLP and Google Scholar. The third dataset linked the products from the product catalogs Abt and Buy [12]. We chose these datasets because they have already been used to evaluate several state-of-the-art machine-learning frameworks for deduplication. In addition, the first two datasets are known to contain data where algorithms which rely on the edit distance can achieve acceptable F-measures. On the other hand, the third dataset, Abt-Buy, is known to be a dataset where the edit distance performs poorly due to the labels of products being written using completely different conventions across vendors. The primary goal of our evaluation was to compare our approach with classifiers based on the generic edit distance. We thus compared our approach with the results achieved by using decision trees and SVMs as implemented in MARLIN [4].

[6] The data used for the evaluation is publicly available at `http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution`

Table 2. Data sets

Source	Size	Target	Size	Correct Links	# Properties
DBLP	2,616	ACM	2,295	2,224	4
DBLP	2,616	Scholar	64,273	5,349	4
Abt	1,081	Buy	1,092	1,098	3

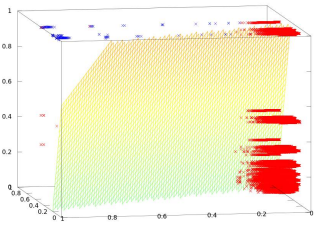
Table 3. Evaluation of ACIDS on three datasets

	DBLP – ACM		DBLP – Scholar		Abt – Buy	
k	5	10	5	10	5	10
F-score (%)	88.98	**97.92**	70.22	**87.85**	0.40	**0.60**
Precision (%)	96.71	96.87	64.73	91.88	0.20	0.30
Recall (%)	82.40	99.00	76.72	84.16	100.00	100.00
Run time (h)	7.53	7.31	8.45	8.36	12.27	12.01
Iterations	2	2	2	2	5	5

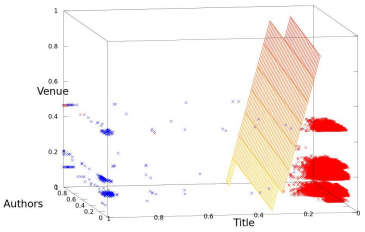
We used the following settings: we used the settings $k = 5, 10$ for the number of examples submitted to the user for labeling. The positive learning rate η^+ was set to 0.1. The boundary for most informative examples Δ was assigned the value 0.1. The maximal number of iterations for the whole algorithm was set to 5. The number of iterations between the perceptron learning and the SVM was set to 50. All experiments were carried out on a 64-bit Linux server with 4 GB of RAM and a 2.5 GHz CPU.

4.2 Results

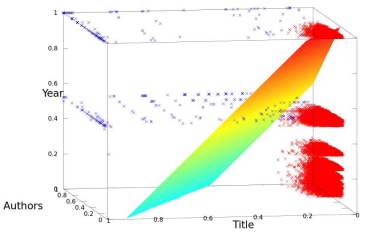
The results of our evaluation are shown in Table 3 and Figure 2. On all datasets, using $k = 10$ led to better results. The termination condition was met on the first two datasets and the classifiers and measures learned outperformed state-of-the-art tools trained with 500 examples (see Fig. 3 in [12]). Especially, on the first data set, we achieved results superior to the 96.4% resp. 97.4% reported for MARLIN(SVM) resp. MARLIN(AD-Tree) when trained with 500 examples. Note that we necessitated solely 40 labeled pairs to achieve these results. In addition, note that we were actually able to outperform all other similarity measures and tools presented in [12] on this particular data set. On the second data set, we outperformed the state of the art (i.e., MARLIN) by more than 7% F-measure while necessitating only 40 labeled pairs. Here both versions of MARLIN achieve less than 80% F-measure when trained with 500 examples. The results on the third data set show the limitations of our approach. Given that the strings used to describe products vary largely across vendors, our approach does not converge within the first five iterations, thus leading to no improvement over the state-of-the art when provided with 100 examples. The main drawback of our approach are the runtimes necessary to compute the most informative examples. A solution to this approach would be to devise an approach that allows for efficiently computing weighted edit distances. Devising such an approach is work in progress.



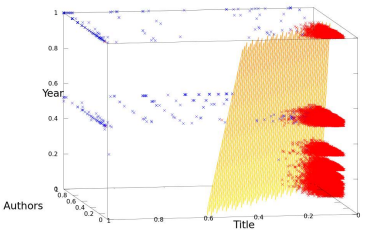
(a) DBLP–ACM dataset with $k = 5$.



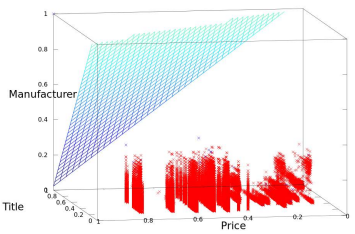
(b) DBLP–ACM dataset with $k = 10$.



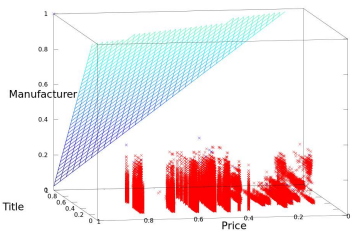
(c) DBLP–Scholar dataset with $k = 5$.



(d) DBLP–Scholar with $k = 10$.



(e) Abt-Buy dataset with $k = 5$.



(f) Abt-Buy dataset with $k = 10$.

Fig. 2. Classifiers learned by ACIDS. Only a subset of all pairs is shown. The red data points are negatives, while the blue are positives.

5 Related Work

Our work is mostly related to Link Discovery and string similarity learning. Several frameworks have been developed with the aim of addressing the quadratic a-priori runtime of Link Discovery. [16,15] presents LIMES, a lossless and hybrid Link Discovery framework, as well as the HYPPO algorithm for the time-efficient computation of links. For example [10] present a lossless approach called MultiBlock that allows to discard a large number of comparisons. Other frameworks include those presented in [21,23]. The second core problem that needs to be addressed while computing links across knowledge bases is the detection of accurate link specifications. The problem has been addressed in manifold ways: The RAVEN approach [17] relies on active learning to learn linear and Boolean classifiers for discovering links. While [11] relies on batch learning and genetic programming to compute link specifications, the approach presented in [18] combines genetic programming and active learning to learn link specifications of high accuracy. In recent work, approaches for the unsupervised computation of link specifications have been devised. For example, [19] shows how link specifications can be computed by optimizing a pseudo-F-measure. Manifold approaches have been developed on string similarity learning (see, e.g., [22,4,3]). [4] for example learns edit distances by employing batch learning and SVMs to record deduplication and points out that domain-specific similarities can improve the quality of classifiers. [3] relies on a theory for good edit distances developed by [2]. An overview of approaches to and applications of learning distances and similarities is given in [9]. To the best of our knowledge, this work is the first to combine active learning and metric learning within the context of link discovery.

6 Conclusion and Future Work

In this paper, we presented an approach for learning string similarities based on edit distances within an active learning setting. We evaluated our approach on three data sets and showed that it yields promising results. Especially, we pointed out that if edit distances are suitable for the properties at hand, then we can actually improve the similarity functions and outperform simple edit distances. Still edit distances are not suitable for use on all datasets. While we focused on how to learn similarity measures based on edit distances, the general idea behind our approach can be easily extended to other string similarity measures. Most of these approaches rely on counting common tokens either in sets or sequences. By replacing the counting of tokens by the addition of weights, we could consequently extend most string similarity measures to weighted measures and learn them within the ACIDS framework. For example, the Jaccard similarity of two strings A and B is defined as $jaccard(A, B) = \frac{|\mathcal{T}(A) \cap \mathcal{T}(B)|}{|\mathcal{T}(A) \cup \mathcal{T}(B)|}$, where $\mathcal{T}(X)$ is the set of tokens (i.e., words) that make up X. Given that all tokens are issued from an alphabet Σ, we can define a weight function $\omega : \Sigma \to [0, 1]$. The weighted Jaccard similarity would then be

$$weightedJaccard(A,B) = \frac{\sum\limits_{x \in (\mathcal{T}(A) \cap \mathcal{T}(B))} \omega(x)}{\sum\limits_{y \in (\mathcal{T}(A) \cup \mathcal{T}(B))} \omega(y)}. \tag{16}$$

We could then apply the ACIDS learning approach to devise an appropriate weight functions ω. Similarly, we can also assign weights to single n-grams derived from any alphabet and learn weighted n-gram-based similarity functions. Thus instead of computing the trigram similarity as $trigrams(A,B) = \frac{2|\mathcal{T}(A) \cap \mathcal{T}(B)|}{|\mathcal{T}(A)|+|\mathcal{T}(B)|}$, (where $\mathcal{T}(X)$ is the set of trigrams that make up X) we would compute

$$weightedTrigrams(A,B) = \frac{2 \sum\limits_{x \in (\mathcal{T}(A) \cap \mathcal{T}(B))} \omega(x)}{\sum\limits_{y \in \mathcal{T}(A)} \omega(y) + \sum\limits_{z \in \mathcal{T}(B)} \omega(z)}. \tag{17}$$

In addition to carrying out exactly this task, we will devise an approach to improve ACIDS' runtime so as to make it utilizable within interactive settings in future work. Moreover, we will evaluate the effect of the initial classifier and learning rate setting on our approach.

References

1. Auer, S., Lehmann, J., Ngonga Ngomo, A.-C.: Introduction to linked data and its lifecycle on the web. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 1–75. Springer, Heidelberg (2011)
2. Balcan, M.-F., Blum, A., Srebro, N.: Improved guarantees for learning via similarity functions. In: COLT, pp. 287–298 (2008)
3. Bellet, A., Habrard, A., Sebban, M.: Learning good edit similarities with generalization guarantees. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011, Part I. LNCS (LNAI), vol. 6911, pp. 188–203. Springer, Heidelberg (2011)
4. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: KDD, pp. 39–48 (2003)
5. Chang, C.-C., Lin, C.-J.: LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2(3), 27:1–27:27 (2011)
6. Cristianini, N., Shawe-Taylor, J.: An introduction to support Vector Machines: and other kernel-based learning methods. Cambridge University Press (2000)
7. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
8. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615 (1987)
9. Hertz, T.: Learning Distance Functions: Algorithms and Applications. PhD thesis, Hebrew University of Jerusalem (2006)
10. Isele, R., Jentzsch, A., Bizer, C.: Efficient Multidimensional Blocking for Link Discovery without losing Recall. In: WebDB (2011)
11. Isele, R., Bizer, C.: Learning linkage rules using genetic programming. In: 6th International Workshop on Ontology Matching, Bonn (2011)

12. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. PVLDB 3(1), 484–493 (2010)
13. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8 (1966)
14. Li, G., Deng, D., Wang, J., Feng, J.: Pass-join: A partition-based method for similarity joins. PVLDB 5(3), 253–264 (2011)
15. Ngonga Ngomo, A.-C.: A time-efficient hybrid approach to link discovery. In: Proceedings of OM@ISWC (2011)
16. Ngonga Ngomo, A.-C., Auer, S.: Limes - a time-efficient approach for large-scale link discovery on the web of data. In: Proceedings of IJCAI (2011)
17. Ngonga Ngomo, A.-C., Lehmann, J., Auer, S., Höffner, K.: RAVEN – Active Learning of Link Specifications. In: Sixth International Ontology Matching Workshop (2011)
18. Ngonga Ngomo, A.-C., Lyko, K.: EAGLE: Efficient active learning of link specifications using genetic programming. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 149–163. Springer, Heidelberg (2012)
19. Nikolov, A., d'Aquin, M., Motta, E.: Unsupervised learning of link discovery configuration. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 119–133. Springer, Heidelberg (2012)
20. Pavel, S., Euzenat, J.: Ontology matching: State of the art and future challenges. IEEE Transactions on Knowledge and Data Engineering 99 (2012)
21. Raimond, Y., Sutton, C., Sandler, M.: Automatic interlinking of music datasets on the semantic web. In: Proceedings of LDoW (2008)
22. Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(5), 522–532 (1998)
23. Scharffe, F., Liu, Y., Zhou, C.: Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In: IK-KR IJCAI Workshop (2009)
24. Settles, B.: Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison (2009)
25. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. Proc. VLDB Endow. 1(1), 933–944 (2008)