Learning Approach for Domain-Independent Linked Data Instance Matching

Khai Nguyen  
University of Science  
Ho Chi Minh, Vietnam  
nhkhai@fit.hcmus.edu.vn

Ryutaro Ichise  
National Institute of Informatics  
Tokyo, Japan  
ichise@nii.ac.jp

Hoai-Bac Le  
University of Science  
Ho Chi Minh, Vietnam  
lhbac@fit.hcmus.edu.vn

ABSTRACT

Because almost all linked data sources are currently published by different providers, interlinking homogeneous instances of these sources is an important problem in data integration. Recently, instance matching has been used to identify owl:sameAs links between linked datasets. Previous approaches primarily use predefined maps of corresponding attributes and most of them are limited to matching in specific domains. In this paper, we propose the LFM, a learning-based instant matching system, which is designed for achieving a reliable domain-independent matcher. First, we compute the similarity vectors between labeled pairs of instances without specifying the meaning of the RDF predicates. Then a learning process is applied to learn a tree classifier for predicting whether the new pairs of instances are identical. Experiments demonstrate that our method achieves a 4% improvement in precision and recall against recent top-ranked matchers, if we use a small amount of labeled data for learning.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database applications - Data mining; I.2.6 [Artificial Intelligence]: Learning; H.3.4 [Information Storage and Retrieval]: Systems and Softwares - Semantic Web

General Terms

Algorithms, Experimentation

Keywords

Linked Data, Entity Resolution, Instance Matching, Supervised Learning.

1. INTRODUCTION

With online data rapidly growing every second, the undeniable benefits of the many current web services (e.g., search engines, data integration, directory organizations) are nevertheless not sufficient to meet the demands for obtaining information from various resources. The year 2006 marked the introduction of linked data\(^1\). With innovative characteristics, especially the linking of data in categories, linked data by itself has helped to solve the online data problem and consequently has attracted much attention. Currently, progress is being made by many linked data construction projects, which range from specific-domain datasets (e.g., geographic places, organizations, celebrities, sciences, medicine, zoology) to cross-domain datasets. According to statistics collected in September 2011, the Linking Open Data (LOD) cloud comprises 31.6 billion Resource Description Framework (RDF) triples existing in 295 datasets\(^2\). This development of linked data brings many new challenges and opportunities to the research community [2].

If we need a lot of information, integrating available datasets is obviously better than constructing a separate new dataset. This information need consequently leads to linked data integration challenges. Among them, instance matching is one of the most important problems. Instance matching in linked data refers to the task of finding identical instances, which co-refer to a real-world object. Since the owl:sameAs predicate is defined to declare identity relationships between instances when publishing linked data, instance matching in linked data has the same meaning as constructing owl:sameAs links. Although linked data from the aspect of information storing are merely data in a richer form [8], the diversity

\(^1\)http://www.w3.org/DesignIssues/LinkedData.html  
\(^2\)http://www4.wiwiss.fu-berlin.de/lodcloud/state
in the representations of an object’s information between datasets forms many barriers. For example, a number of vocabularies (e.g., dc, foaf, skos, geo) currently exist. However, two primary problems require attention. First, linked data publishers use different vocabularies because of the requirements of specific domains. Second, the information of an object described in each data source is also different. Let us explain the problems by the example shown in Figure 1, in which owl:sameAs links two identical instances referring to Harlem. In NYTimes, the place name is Harlem (NYC), and the geographic coordinates are provided within the instance. In Freebase, the place name is given in various languages, for example, Harlem in English and Harlemas in Lithuanian, and the geographic coordinates are provided in another linked instance (m.0clyh58). It is also more difficult when the vocabularies used in each data sources differ. While the second problem of different information causes major difficulties for instance matching, the first problem of different vocabularies is closely related to problems of domain-free matching.

In recent years, many efforts have been made in the automatic construction of owl:sameAs links for linked data. The basic method is to measure the similarity level between instances by using the information provided in the RDF triples. Then, a threshold-based decision step is applied to determine whether two instances match. Most previous approaches share a common limiting point. These approaches concentrate on a particular domain of data or utilized manual definitions of predicate pairs [3, 6, 12, 13]. Consequently, because real linked datasets are very diverse, these methods encounter many difficulties in connecting different pairs of undefined data sources. Even now, proposals for domain-free instance matching [10, 14, 15] still rely on manual settings.

In our work, we are interested in developing a novel system that can perform instance matching tasks for datasets from various domains. Since a learning process can automatically adapt a matching model to learned datasets, we propose a machine learning based method for instance matching, which does not depend on domain of datasets. We call our system LFM (learning-based domain-free matcher). First, we compute the similarity vectors between labeled instances by using object’s value of RDF triples. Then these vectors are used to train a decision tree classifier. This classifier not only works well for domain-independent matching but also is easily analyzed for developing further cross-domain matchers. We test our system on real datasets and provide some experimental results, which verify the desirable performance of our system. We also compare this approach with the most recently developed matchers on the same datasets and show that our system improves accuracy and recall.

This paper is organized as follows. In Section 2, we summarize previous studies in the area of discussion. Next, we present our proposed system, LFM, in detail in Section 3 and give the experimental results in Section 4. Finally, we state our conclusions and future work in Section 5.

2. RELATED WORK
We separate current literature into two main groups: matchers that use pre-defined corresponding predicates [16, 6, 12, 13, 14] and matchers that do not need this information [1, 15, 10].

Among the first group, Silk [16] provides both an instance matching framework with a declarative language to discover links and various metrics for each data type and some combination methods to aggregate similarities. Silk searched for owl:sameAs links between cities instances from DBpedia and Geonames with 87% recall. However, when using this framework, the user must define the properties of each instance to be compared. In addition, two thresholds for acceptance and verification need to be configured. LinkedMDB [6] matches instances in the movie domain, which includes directors, actors, producers, and films. This system aims to discover high-quality links between LinkedMDB and many other linked databases (e.g., DBpedia, YAGO, Geonames, MusicBrainz). Like Silk, LinkedMDB uses multiple similarity functions such as edit distance, HMM-based distance, Jaccard, cosine, and BM25. The limitation of this approach is that the main feature used in matching identical instances is just the name of objects. Zhishi Links [12] performs two-step matching. First, ZhishiLinks uses the label of objects provided in resources through known predicates, such as skos:prefLabel or scheme:label, to remove instances that are highly different. In the second phase, a more complex metric at a semantic level is utilized. This system was ranked first in the OAEI 2011 instance matching track and is one of the current state-of-the-art matchers. AgreementMaker [3] is also a typical system. The approach of this system consists of three steps. The first step is to generate the candidates set by comparing the labels of instances. The second step is the disambiguation phase, which divides the candidates into smaller subsets. The final step is matching every pair of instances in each subset to obtain the final alignment. We compare our approach with both Zhishi Links and AgreementMaker in our experiments. Another example of similarity-based instance matching algorithms is proposed in [13]. The primary idea of that algorithm is to use the relationships between objects (songs, authors, singers, etc.) to build a graph. The underlying assumption is that using the related information of instances is better than comparing individual pairs of instances. A matching method based on graph matching was implemented to connect Jamendo and MusicBrainz. Scharffe et al. [14] proposed RDF-AI, which contains RDF datasets matching. This system considers instances as graphs, in which the nodes are linked instances and their properties. Then the matching phase becomes a graph matching technique. To estimate the correlation of two instances, RDF-AI chooses the highest value of similarities between all pairs of interlinked resources. The same method is also used to compute the similarity of two resources according to their properties. Thus, this system tends to lean towards positive cases and may accept many impostors. In general, the above approaches compare two instances by their specified properties. That is, the false rejection rate is minimized. However, these systems limit the information stored in instances, which may lose precision in matching highly ambiguous datasets. Moreover, it is difficult to apply these systems for matching undefined data sources or vocabularies. Therefore, these systems are de-
signed to focus on a particular domain, whereas approaches in the second group can be adapted to match various domains.

Among the second group, SERIMI [1] automatically extracts frequent predicates as the main properties of objects. SERIMI chooses the best correlated pair between data sources and compares it with a matching threshold. This system focuses only on short strings, which usually refer to an object’s name or other short description. Recently, Silk has been implemented a method for learning linkage rules [7]. The authors use genetic algorithm for optimizing the rules. This work makes Silk achieves a schema-independent matcher. Song et al. [15] focused on blocking scheme for data interlinking. The author proposed an interesting technique that automatically selects the most important predicates of each data source. They use the harmonic mean of discrimination and the coverage of predicates. The idea of this study can be used for achieving a schema-independent instance matching system. Le et al. [10] proposed an interlinking method that can discover links between multiple given data sources. The authors assume that homogeneous instances link to each other in common ways. From known owl:sameAs links, frequent graphs are extracted to construct pattern templates. Since data sources may contain many unlinked identical instances, the templates are used as guidelines to discover hidden relationships. Instance similarity computation does not depend on the data domain, and so this method can be extended for matching various domains of data without a predefined schema. Nevertheless, if the links between data sources in advance are not sufficient, this method fails to retrieve good frequent patterns.

Compared with previous studies, our approach differs in instance similarity estimation and the learning model. Like Silk, we separate similarity functions for various data types, but we utilize different measurements and retain more detailed information of the similarities for the learning process. Like Song et al. [15], we find the most helpful predicates for matching, but in implicit ways. Our method is based on supervised learning for automatically producing a decision tree. Thus, manual settings like those in previous approaches are minimized. In addition, we use the advantage of linked data, which is the related information of instances given in their links. This advantage was not applied in previous approaches.

3. LEARNING-BASED DOMAIN-FREE MATCHER

3.1 Overview

We describe the LFM (learning-based domain-free matcher) through two phases: the learning phase for determining a classifier from the labeled data and the predicting phase for matching the status for new pairs of instances. The general framework of the LFM is shown in Figure 2. To find identical instances between the source dataset $D_s$ and the target dataset $D_t$, we assume that a few identical instances have already been labeled in advance. In the training phase, we apply a predicate pre-filtering step from both labeled and unlabeled data to remove useless RDF predicates, which do not describe important information. The useful predicates are used to collect information of the instances. After that, we apply a similarity computation step to construct the similarity vectors between labeled pairs of instances for positive and negative cases. Then, the vectors are fed into the classifier learning step. The result of the classifier learning step is a binary classifier, which predicts positive (matching) and negative (not matching) cases. In the prediction phase, the similarity vectors of given instances are extracted by the similarity computation step. Then, their similarity vectors are passed through the binary classifier to determine whether they match. Matching for all instances between the source dataset and the target dataset is performed in the same way. We use the classifier to predict the matching status for each possible pair of instances.

Linked data allows instances to contain related information in object value of RDF triples (as illustrated in Figure 1). In many cases, full information of an object can be retrieved sufficiently by only following the linked Uniform Resource Identifiers (URIs). Therefore, unlike previous methods, we utilize all the information given in not only an instance itself, but also in direct linked instances if they exist. Suppose that $<s,p,o>$ represents an RDF triple (subject, predicate, object); then, an instance $x$ has a set of RDF triples: $T(x) = \{<s_x,p_x,o_x>\}$. Since we also use information given in referred resources from any instance of interest, instance $x$ is extended to

$$E_x(x) = T(x) \cup \bigcup_{<s_x,p_x,o_x> \in T(x)} \{T(y)|y = o_x\}$$  \hspace{1cm} (1)

where $o_x$ refers to a linked data resource.

In our study, only directly linked resources in the same data source of instances of interest are used. That means instance $y$ is also stored in same data sources as $x$. In many cases, a predicate may be used more than one time in an instance. From that viewpoint, we group triples sharing the same predicate into one triple with multiple of object’s values. Consequently, $T(x)$ becomes a set of $<s_x,p_x,V_x>$, where $V_x$ is the set of object’s values described by the same predicate. By applying (1) and grouping object’s values having a similar predicate, we refer to $E_x(x)$ as an extended instance of $x$.

In the rest of this section, we describe details of each LFM step, which consists of predicate pre-filtering, similarity computation and classifier learning.
3.2 Predicate Pre-filtering

The goal of this step is to filter useful predicates from all predicates used in a whole data source. We apply a statistic-based method: the frequency of a predicate is used to determine whether it is important. We formalize \( f(p_i) \) as the frequency of predicate \( p_i \) that exists in instances of dataset \( D_i \), as follows:

\[
 f(p_i) = \frac{|\{x \in D_i \mid \exists s_x, o_x \in x, p_x = p_i\}|}{|\{x \in D_i \mid \exists s_x, o_x \in x, p_x = p_i\}|} \arg \max_{p_j \in D_j} \frac{|\{y \in D_j \mid \exists s_y, o_y \in y, p_y = p_j\}|}{|\{y \in D_j \mid \exists s_y, o_y \in y, p_y = p_j\}|} 
\] (2)

Then, only predicates \( p_i \) whose \( f(p_i) \) is greater than a threshold \( \alpha \) are selected. Choosing this threshold is very important: \( \alpha \) should be large enough to remove useless predicates but small enough to retain useful predicates. The results of this step are two lists of useful predicates, \( P_{Ds} \) and \( P_{Dt} \), for the source dataset and the target dataset, respectively. These lists are used as templates for extracting the information of extended instances. This information is the input of the next step.

This task is important because the diversity of RDF predicates used in linked data is very high. Moreover, because our system is learning-based, too much noisy information makes it difficult to learn an efficient classifier. In addition, the decrease of predicate heterogeneity minimizes the computation cost. This method works because each domain has specific characteristics, which can be described as a common set of predicates. For example, 80,000 random instances in the geography domain of DBpedia have 8365 distinct predicates, although only 94 predicates have a frequency over 10%. Therefore, to remove useless predicates, we use the frequency of predicates to estimate its importance.

In a comparison of our method with other methods, Le et al. [10] do not separate useless and useful predicates explicitly. They rank predicates according to their frequency to adjust the contribution of each predicate. Our method is slightly similar to this approach but different in the frequency computation. Song et al. [15] utilize a predicate selection step. By traversing all instances in a dataset, the discrimination factor and the coverage factor of each predicate are computed. Then these factors are compared with bounding thresholds to decide whether the predicate is important. Our method is more simple than this approach, because we also apply supervised learning, which has the ability to implicitly recognize which information is important.

3.3 Similarity Computation

After extracting the information from each instance of a pair of interest, the next step is to compute the similarity between them. In our system, each predicate pair of two instances forms a similarity. Therefore, by collecting all similarities, we can build a similarity vector. In addition, our system utilizes the classification of the data type, which is also discussed in this section.

The similarity vector between two given instances \( x \) and \( y \) is computed from extended instances \( E_s(x) \) and \( E_s(y) \), combined with pre-filtered predicates of each data source. These predicates are used in collecting information of the instances. In other words, predicate \( p_i \) is collected only if it exists in both \( E_s(x) \) and the list of frequent predicates in a dataset containing \( x \). This process is the same for collecting information of \( y \). Because a predicate may not be used in all RDF triples of an instance, missing values may occur when collecting the information using frequent predicates. This issue leads to a variety of dimensions of the similarity vector between different pairs. To unify these vectors, we align all vectors into the most general case, which is the vector produced by the combination of the frequent predicate lists of the source dataset and the target dataset. We formulate this process by the example shown in Figure 3. Suppose that we need to construct the similarity vector for two data sources: \( D_s \) and \( D_t \). Let \( P_{Ds}, P_{Dt} \) be the frequent predicate lists and \( U_{Ds}, U_{Dt} \) be the sets of values of the instances. Our method combine predicates whose object’s values have the same data type to generate an element of the similarity vector. Because \( U_{Ds} \) and \( U_{Dt} \) are the sets of multiple values, we need to choose representative values for similarity.

To do this, we compute the similarities between all object’s values of the selected predicate pairs and collect the maximum, minimum, and average values of these similarities. Therefore, the dimension of vectors is always less than or equal to \( 3 \times |P_{Ds}| \times |P_{Dt}| \). In Figure 3, we can obtain three similarities \( S_0, S_1, \) and \( S_2 \) from \( p_0 \) in \( P_s \) and \( p_6 \) in \( P_y \).

Computing similarity for two values, we perform different similarity measurement for various data types of values. In our study, we classify data into 4 types: string, URI, numeric and date-time.

1. **String**, we use a combination of the token-based **Word Overlap** (normalized version [11]) and the lexical-based **Jaro** [9]. We formalize the similarity function between two texts as follows:

\[
sim(X,Y) = \sum_{x \in X, y \in Y} \frac{\text{Jaro}(x,y)}{\log_2 |X| + \log_2 |Y|}.
\] (3)

Since instance matching is primarily related to matching names and descriptions of instances, which are often provided in linked data, the matchers should implement functions that are appropriate for estimating the similarity of short strings (e.g., names) and longer strings (e.g., descriptions). Therefore, we compute the similarity between two strings at the word level.
2. **URI.** Many previous approaches miss the information of URIs. In Silk [16], the comparer simply returns 1 or 0 for URI strings totally matched or not totally matched, respectively. That means two URIs are matched only if they refer to the same resource, even though an object’s information is often stored in various resources. In our study, if a URI does not refer to linked data resources in the same data source, we dereference URI to retrieve the web page title. If the URI is not dereferenceable, we assume that the object’s information is stored in the last two levels of the URI. For example, http://www.example.com/resource/computer/mouse is reduced into a string with two words: computer and mouse. From now on, we apply the string similarity measurement for these processed URIs.

3. **Numeric and Date-time.** For numeric and date-time, we first compute the absolute value of the difference. Then, to synchronize with the correlation metric of the string type (the higher value, the higher similarity), we take the reciprocal of this value. In addition, we separate floating-point and integer into two different data types.

To determine the data type for $V_s$, we use the most frequent data type of all elements in $V_s$, because RDF objects may have various data types, not except for object’s values of the same predicate. However, if we focus on a particular domain, these values usually receive the same data type. For example, we extracted the data type of the predicate http://DBpedia.org/property/birthPlace from 40,000 random instances in eight different domains of data in DBpedia. The ratio of the number of string values to URI values is 4554:745. If we extract object’s values from 5000 instances in the people domain, this ratio significantly increases to 4203:67, meaning the string is more frequently used to describe the birthplace than are other data types.

Most previous approaches consider all different data types as string [10, 14, 15]. This is not appropriate in domain-free matching, because various domains have specific common characteristics. For example, if given datasets belong to music album, the important content exists as a string (e.g., name, artists, composer) and date-time (e.g., release date), while in the geography domain important information is described by a string (name) and a double (geographic coordinates). In addition, the ambiguity level in large-scale datasets is much higher. Therefore, we must pay attention to the helpfulness of the data type.

### 3.4 Classifier Learning

We use the matching label of each pair of instances as the value for the class in the training set. As illustrated in Table 1, each feature vector has a similarity value and a class label for matching the status of a corresponding pair. As mentioned in the discussion of the effect of threshold $\alpha$ in Section 3.2, if we reserve all useful predicates, noisy predicates may also be chosen. In this case, similarity vectors may contain attributes whose values are almost missing. Moreover, non-related predicates between data sources may occur with a high frequency. This means a large number of mostly zero attributes are present. Therefore, we define $\beta$ to remove the attributes that have a ratio of non-missing or non-zero values under this threshold. For example, if attribute $\min\{P_{D_3} \leftrightarrow P_{D_1} \leftrightarrow P_{D_0}\}$ in Table 1 remains almost missing for all other instances, it should be eliminated.

We implement the decision tree classifier for our system. The training set is very imbalanced because the majority of cases are negative, because the training set is constructed from labeled matching pairs. Therefore, a probability-based classifier may not perform well whereas a decision tree still works. On the other hand, if we extend our system to a large-scale instance matchers, the useful features should be exactly identified. Every classification model always contains information about the importance of each feature. As the representation of decision tree, a feature that is used in classification is clearly stated. If we extract the combined predicates of these features, it is easy to determine which predicates of source data should be map to which predicates of target data. The matching stage therefore be more faster because the information needed to be processed is reduced. In addition, the imbalance can be solved by a simple over-sampling method that applies random sampling on non-matched pairs. Although in most problems, random sampling leads to over-fitting of the data, using this method to match linked data instances is feasible, because matched pairs or non-matched pairs share many common characteristics. This means the learning process does not need too much data and random sampling can be used to overcome the problem of imbalance. Our experiments focus on these issues.

### 4. EXPERIMENTAL EVALUATIONS

#### 4.1 Experimental Purpose and Datasets

We evaluate our system from aspects closely related to all of the problems stated:

1. Matching on various domains.
2. Over-sampling for resolving the imbalance problem.
3. Utilizing a small percentage of ground truth.

For (1), we use nine real datasets. The first one is the DBpedia$\leftrightarrow$GeoNames dataset (D1), which belongs to the Geography domain and consists of 86,547 pairs of identical instances referring to a variety of places. The second dataset is the movie domain: DBpedia$\leftrightarrow$LinkedMDB (D2). Objects referred by instances in this dataset are about films, actors, directors, and producers. To compare our methods with some of the most recent approaches, we select datasets of the OAEI 2011 Instance Matching track for the seven remaining datasets (D3-D9). These datasets contain owl:sameAs links between NYTtimes and DBpedia, Freebase, and Geonames. The OAEI 2011 datasets have three topics: Location, Organization, People as the domains of real-world objects. The predicates of different domains are not the same, even when these domains belong to the same data sources. Therefore, although in selected datasets, D3 $\rightarrow$ D9 involve with NYTtimes and many instances involve with DBpedia, the tested datasets are not bias for a particular
Table 1: Partial view of the training data for the learning classifier.

<table>
<thead>
<tr>
<th>( \max {P_{Ds,p_0} \leftrightarrow P_{Dt,p'_1}} )</th>
<th>( \text{avg}{P_{Ds,p_0} \leftrightarrow P_{Dt,p'_1}} )</th>
<th>( \min {P_{Ds,p_2} \leftrightarrow P_{Dt,p'_0}} )</th>
<th>( \min {P_{Ds,p_2} \leftrightarrow P_{Dt,p'_0}} )</th>
<th>Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.176</td>
<td>0.100</td>
<td>0.001</td>
<td>0.012</td>
<td>True</td>
</tr>
<tr>
<td>?</td>
<td>0.425</td>
<td>?</td>
<td>?</td>
<td>True</td>
</tr>
<tr>
<td>0.501</td>
<td>0.981</td>
<td>?</td>
<td>0.123</td>
<td>False</td>
</tr>
<tr>
<td>0.001</td>
<td>0.296</td>
<td>?</td>
<td>0.981</td>
<td>False</td>
</tr>
<tr>
<td>0.712</td>
<td>?</td>
<td>?</td>
<td>0.233</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 2: Overview of the datasets. The number of RDF triples is counted from instances interlinked by owl:sameAs links and referred instances.

<table>
<thead>
<tr>
<th>ID</th>
<th>Source((D_s))</th>
<th>Target((D_t))</th>
<th>Domain</th>
<th>RDF Triples</th>
<th>Identities count</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>DBpedia</td>
<td>Geonames</td>
<td>Geography</td>
<td>2416181</td>
<td>86547</td>
</tr>
<tr>
<td>D2</td>
<td>DBpedia</td>
<td>LinkedMDB</td>
<td>Movie</td>
<td>1202344</td>
<td>13795</td>
</tr>
<tr>
<td>D3</td>
<td>NYTtimes</td>
<td>DBpedia</td>
<td>Geography</td>
<td>312212</td>
<td>1920</td>
</tr>
<tr>
<td>D4</td>
<td>NYTtimes</td>
<td>DBpedia</td>
<td>Organization</td>
<td>210818</td>
<td>1965</td>
</tr>
<tr>
<td>D5</td>
<td>NYTtimes</td>
<td>DBpedia</td>
<td>People</td>
<td>618893</td>
<td>4977</td>
</tr>
<tr>
<td>D6</td>
<td>NYTtimes</td>
<td>Freebase</td>
<td>Geography</td>
<td>559578</td>
<td>1920</td>
</tr>
<tr>
<td>D7</td>
<td>NYTtimes</td>
<td>Freebase</td>
<td>Organization</td>
<td>707761</td>
<td>3044</td>
</tr>
<tr>
<td>D8</td>
<td>NYTtimes</td>
<td>Freebase</td>
<td>People</td>
<td>1486652</td>
<td>4979</td>
</tr>
<tr>
<td>D9</td>
<td>NYTtimes</td>
<td>Geonames</td>
<td>Geography</td>
<td>163293</td>
<td>1920</td>
</tr>
</tbody>
</table>

Figure 4: Distribution of predicates occurring at frequency [0, 0.1, 0.1] in DBpedia of dataset D1 (DBpedia↔Geonames).

Table 3: Number of predicates and feature vector dimensions. \(D_s\) and \(D_t\) are the source dataset and target dataset, respectively. \(P_{Ds}\) and \(P_{Dt}\) are the sets of distinct predicates in \(D_s\) and \(D_t\), respectively. \(P'_{Ds}\) and \(P'_{Dt}\) are the predicate sets filtered by applying threshold \(\alpha\). \(|T|\) is the dimension of the vector by combining \(P_{Ds} \times P_{Dt}\). \(T'\) is the reduction of \(T\), after removing almost-missing and almost-zero attributes by the threshold \(\beta\).

| Dataset | \(|P_{Ds}|\) | \(|P_{Dt}|\) | \(|P'_{Ds}|\) | \(|P'_{Dt}|\) | \(|T|\) | \(|T'|\) |
|---|---|---|---|---|---|---|
| D1 | 8377 | 17 | 58 | 14 | 630 | 104 |
| D2 | 1360 | 54 | 49 | 15 | 906 | 102 |
| D3 | 12 | 2089 | 11 | 89 | 525 | 151 |
| D4 | 10 | 1767 | 10 | 38 | 306 | 56 |
| D5 | 10 | 1966 | 10 | 38 | 279 | 72 |
| D6 | 12 | 781 | 11 | 32 | 246 | 34 |
| D7 | 10 | 1496 | 10 | 51 | 246 | 24 |
| D8 | 10 | 1846 | 10 | 49 | 246 | 21 |
| D9 | 12 | 32 | 11 | 28 | 201 | 33 |

To determine the threshold \(\alpha\), we conducted a survey on the nine datasets. We provide the statistics for the number of predicates of DBpedia in the dataset DBpedia↔Geonames in Figure 4. At small frequencies, the large number of distinct predicates in DBpedia indicates that the heterogeneity of predicates in this source is very high. The number of predicates decreases along with their frequency and remains stable at a 20% frequency. The stable distribution of predicates in all the remaining datasets is approximately equal to or higher than this percentage. Therefore, we set the threshold \(\alpha\) to 0.2 for all evaluations. In addition, we use the value 0.1 for the threshold \(\beta\) mentioned in Section 3.4. By using these thresholds, the number of predicates and the feature vector dimensions are as reported in Table 3. We conducted all evaluations using the WEKA[5] J48 decision tree.  

data source in the context of domain-independent. More detailed information about the experimental datasets is given in Table 2. In addition, for constructing the testing dataset, we dereferenced the URIs of the resources to retrieve RDF files. Of course, all triples containing owl:sameAs and rdfs:seeAlso predicates (usually used with the same meaning as owl:sameAs) have been removed. For (2), we apply non-replacement random over-sampling on each dataset to select an equal number of negative cases in order to have a 50%:50% balanced set for evaluation. For (3), we perform experiments at different splits of training and testing sets to evaluate our method. For each training split trial, we randomly separate an over-sampled dataset into the training set and the testing set. This process is repeated 100 times to obtain 100 different results. Finally, the bootstrapping method is applied to evaluate the confidence of all results.
Table 4: Precision and recall for evaluation datasets with different splits of training data (99% confidence interval).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>10%</th>
<th>5%</th>
<th>2%</th>
<th>1%</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>0.98±0.00034</td>
<td>0.98±0.00032</td>
<td>0.98±0.00033</td>
<td>0.98±0.00034</td>
<td>0.98±0.000895</td>
</tr>
<tr>
<td>D2</td>
<td>0.99±0.00044</td>
<td>0.99±0.00042</td>
<td>0.99±0.00041</td>
<td>0.99±0.00043</td>
<td>0.98±0.00018</td>
</tr>
<tr>
<td>D3</td>
<td>0.98±0.00137</td>
<td>0.97±0.00303</td>
<td>0.97±0.00276</td>
<td>0.94±0.000638</td>
<td>0.96±0.00381</td>
</tr>
<tr>
<td>D4</td>
<td>0.96±0.00086</td>
<td>0.96±0.00133</td>
<td>0.95±0.00392</td>
<td>0.95±0.00783</td>
<td>0.96±0.00914</td>
</tr>
<tr>
<td>D5</td>
<td>0.99±0.00100</td>
<td>0.99±0.00113</td>
<td>0.98±0.00146</td>
<td>0.98±0.00170</td>
<td>0.98±0.00092</td>
</tr>
<tr>
<td>D6</td>
<td>0.99±0.00062</td>
<td>0.99±0.00085</td>
<td>0.99±0.00126</td>
<td>0.98±0.00141</td>
<td>0.99±0.00317</td>
</tr>
<tr>
<td>D7</td>
<td>0.98±0.00066</td>
<td>0.98±0.00093</td>
<td>0.97±0.00121</td>
<td>0.96±0.00329</td>
<td>0.96±0.00028</td>
</tr>
<tr>
<td>D8</td>
<td>0.99±0.00042</td>
<td>0.99±0.00047</td>
<td>0.99±0.00073</td>
<td>0.99±0.00110</td>
<td>0.99±0.00012</td>
</tr>
<tr>
<td>D9</td>
<td>0.99±0.00072</td>
<td>0.99±0.00186</td>
<td>0.97±0.00309</td>
<td>0.96±0.00305</td>
<td>0.97±0.00098</td>
</tr>
</tbody>
</table>

4.2 Experimental Results

4.2.1 Results on different data splits.

This section reports detailed results on the precision and the recall over all experimental datasets. We tested the LFM at different levels of training splits. Table 4 describes the results for representative splits at 10%, 5%, 2%, and 1% and 50 labeled pairs used for learning. The maximum difference of the precision and recall is very small when comparing the sizes of all the training splits. The maximum difference is only 4% for precision and 5% for recall. This means our system can achieve comparable precision and recall even with a small training set. Therefore, our approach can be used for matching data sources with only a small set of labeled data required. Furthermore, at the high confidence of 99%, the error rates have narrow bounds, which average 0.189% and 0.191% for precision and recall, respectively. This confirms that random over-sampling, which is implemented in our system, is appropriate for resolving the imbalance problem of supervised learning in linked data instance matching.

4.2.2 Comparison to previous systems

We compare our system with SERIMI [1], AgreementMaker [3] and Zhishi.Links [12], which joined the recent OAEI 2011 instance matching track. In this track, participants were asked to interlink identical instances from NYTimes to DBpedia, Freebase, and Geonames. These datasets were also used in our evaluation (D3→D9), as previously presented. Since these systems are non-learning based matchers, we used the result for the 1% training split for comparison. The number of labeled pairs used for learning ranged from 18 to 50. Details of the comparison are shown in Table 5. The comparison results confirm that when using a small number of labeled data for training, our system highly improves the precision, the recall and the f-measure over all datasets. AgreementMaker has comparable precision with the LFM on dataset D5 but it fails in recall and therefore also in f-measure. The Zhishi.Links results nearly reach the LFM results in dataset D5 but overall the LFM performs much more accurately than this state-of-the-art system. We presume that the previous systems manually choose thresholds for matching, which do not rely on the characteristics of the data sources, and so they could not gain the highest results.

5. CONCLUSION

In this paper, we introduced the LFM, a domain-free linked data instance matching system. Compared with previous approaches, we use more information, which is provided in related resources of instances of interest. We also apply various similarity functions for different data types. Our method, which relies on supervised learning, can learn an effective decision tree from a small set of labeled data. The experimental results demonstrate that the LFM can achieve high precision and high recall in matching real datasets. When using a small amount of ground truth, our system improves matching precision and recall by 4% against the recent best instance matchers. Moreover, we confirm that random sampling is appropriate for resolving the imbalance issue when utilizing learning-based methods for linked data instance matching problem.

Although our system is highly accurate in tested datasets, we need to improve the similarity computation for strings by using other semantic-based metrics. Moreover, we did not apply a predicate post-filtering step by exploiting the decision tree. By doing this, the predicting phase can perform faster and avoid computing similarities between useless predicates. In addition, our datasets in the experiments are not large enough to represent large-scale datasets. We are interested in removing manually parameters $\alpha$ and $\beta$ to ob-
tain a fully automatic instance matchers. Beside, we will study on unsupervised approaches to learn the corresponding predicates of two data sources. We will also investigate blocking methods to reduce the matching pool for large-scale datasets. Finally, experiments on much larger datasets would be conducted, such as the 7 million identical resources belonging to various domains between DBpedia ↔ Freebase and Billion Triple Challenges.

6. REFERENCES


