A Machine Learning Approach for Instance Matching Based on Similarity Metrics

Shu Rong[1], Xing Niu[1], Evan Wei Xiang[2], Haofen Wang[1],
Qiang Yang[2], and Yong Yu[1]

[1] APEX Data & Knowledge Management Lab, Shanghai Jiao Tong University
{rongshu,xingniu,whfcarter,yyu}@apex.sjtu.edu.cn
[2] Department of Computer Science and Engineering,
Hong Kong University of Science and Technology, Hong Kong
{wxiang,qyang}@cse.ust.hk

Abstract. The Linking Open Data (LOD) project is an ongoing effort to construct a global data space, i.e. the Web of Data. One important part of this project is to establish `owl:sameAs` links among structured data sources. Such links indicate equivalent instances that refer to the same real-world object. The problem of discovering `owl:sameAs` links between pairwise data sources is called *instance matching*. Most of the existing approaches addressing this problem rely on the quality of prior schema matching, which is not always good enough in the LOD scenario. In this paper, we propose a schema-independent instance-pair similarity metric based on several general descriptive features. We transform the instance matching problem to the binary classification problem and solve it by machine learning algorithms. Furthermore, we employ some transfer learning methods to utilize the existing `owl:sameAs` links in LOD to reduce the demand for labeled data. We carry out experiments on some datasets of OAEI2010. The results show that our method performs well on real-world LOD data and outperforms the participants of OAEI2010.

Keywords: Linking Open Data, instance matching, similarity matric, machine learning, transfer learning.

1 Introduction

Linked Data[4] is a way to construct a global data space, the Web of Data, by interconnecting many structured data sources within the Linking Open Data[1] (LOD) project. These data sources are published under the Resource Description Framework[2](RDF). Each of them may contain millions of RDF triples.

The main idea of Linked Data is to construct typed links between different data sources. Such links describe the relationships between things so that users can browse data among sources by navigating along the links and agents can provide expressive query capabilities over the data on the web just like a local

---

[1] http://linkeddata.org/

[2] http://www.w3.org/RDF/

database. An important link is `owl:sameAs`[3] which indicates that the two instances it links refer to the same real-world object. Different data sources may have different emphases in describing things. Various descriptions can be aggregated according to the `owl:sameAs` links.

Currently, there are more than 300 data sources in LOD while there were only twelve of them in 2007 when the project started. As more and more data sources emerge, there is an urgent demand to provide `owl:sameAs` links from the new data sources to the existing ones. At the same time, the existing links in LOD are not as extensive as one would hope[17]. Instance matching is a practical idea for constructing such links. In general, the existing approaches for matching instances in LOD can be divided into two types. One is based on rules and the other is based on the similarity metric of instance pairs. Most of these methods do not always work well in the LOD scenario since they depend on the result of property matchings. Property matching links the properties from different data sources which have similar semantics such as `foaf:name` and `dc:title`. Property matching is not trivial since the data sources usually design their own ontologies to describe things. Furthermore, we noticed that although some properties in heterogeneous ontologies can not match, they have some connotative relationships. Their values may be worth considering for instance matching. For example, Freebase[4] says that the `fb:profession` of Steve Jobs is "Chief Executive Officer" and his `fb:place_of_death` is "Palo Alto", whereas DBpedia[5] says his `dbp:residence` is "Palo Alto California" and the information about "Chief Executive Officer" is in the text of the `dbp:abstract`. Such information will be ignored in property matching based methods, although it could be significant for human beings to judge whether the two Jobses match. We are inspired to explore the "common-sense " used for matching instances. The goal of this paper is to develop an automated instance matching method that is "common" and provides high accuracy. Such method should be independent of property matching to achieve "commonality".

In this paper, we employ machine learning models for instance matching based on some similarity metrics of instances. The matching instance pairs may have some common features in the similarity metrics of each pair. For example, two matching instances may share some significant words such as "Palo Alto" and "Chief Executive Officer" which we mentioned above, while the non-matching ones may not. Sharing some significant words is a common feature of the matching instance pairs here. We design a similarity vector independent of property matching to represent such features. Based on this vector, we train a learning model to classify the instance pairs as matching or non-matching. To minimize the demand for training data and promote the performance, we try to use existing instance matching information in LOD for help. A transfer learning method is applied to implement this idea.

---

[3] `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/#owl_sameAs/`

[4] `http://www.freebase.com/`

[5] `http://www.dbpedia.org/`

We tried our approach on real LOD data sources which were chosen for IM@OAEI2010[6]. Our performance is better than the participating teams'. Another comparative experiment shows that the existing matching information can really help matching instances from the new data source pairs.

The following technical contributions are made:

- We utilize the values of non-matching properties for instance matching. Such values can be useful but are usually ignored by the existing instance matching approaches.
- We propose a novel approach for instance matching which is independent of property matching with high accuracy.
- We use existing `owl:sameAs` links to help match instances. Due to the heterogeneous ontologies constructed by various data sources, such information is hardly utilized in the existing instance matching approaches.

The remainder of this paper is structured as follows. Section 2 gives some definitions about instance matching and an overview of our proposed approach. Section 3 describes the feature extraction process. The selection of machine learning models is discussed in Section 4. Experimental results on the LOD datasets from LOD are reported in Section 5. Some related work is discussed in Section 6. Finally, Section 7 concludes this paper and discusses future work.

2 Definition and Framework

2.1 Problem Definition

An instance consists of some property-value descriptions about a real-world object. A pair of distinct instances a and b match if they refer to the same object, denoted by $(a, b) \in \mathcal{R}$. In LOD, `owl:sameAs` links are established between matching instances. When establishing such links, we usually consider a pair of data sources each time.

Definition 1 (Instance Matching). *Given two data sources A and B as input, the goal of instance matching is to compute the set $\mathcal{M} = \{(a, b) | (a, b) \in A \times B, (a, b) \in \mathcal{R}\}$.*

According to the definition of instance matching, the problem of finding matching instance pairs can be formalized as a binary classification problem.

Definition 2 (Instance Matching as Binary Classification). *Given two data sources A and B, the goal of instance matching is to find a classifier $\mathcal{C}$: $(a, b) \to \{-1, 1\}$ for $(a, b) \in A \times B$ such that $\mathcal{C}$ maps the non-matching instance pairs to class -1 and the matching ones to the class 1.*

The binary classification problem can be solved by traditional machine learning algorithms, which require multidimensional features as the input. In the problem of instance matching, we extract a feature vector from each instance pair (a, b).
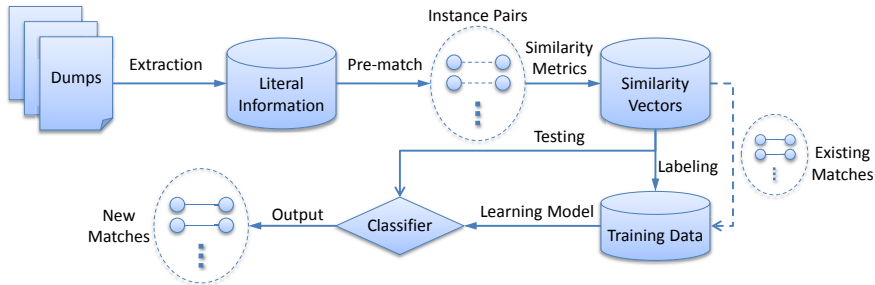
---

[6] `http://oaei.ontologymatching.org/2010/im/index.html`

Fig. 1. Overview of the framework

Definition 3 (Similarity Vector of Instance Pairs). *The n-dimensional feature vector v of an instance pair (a, b) consists of n various similarities of instance a and b. Dimension $v_i = d_i(a, b)$, where d_i is the ith similarity metric function for (a, b).*

The feature vector of an instance pair indicates the similarities of the two instances, which are computed by several metric functions. Some existing instance matching (record linkage[11]) approaches also extract such a feature vector for classification, while each d_i computes the similarity of two values, one from each instance, that belong to a pair of matching properties (fields). Unlike these approaches, our similarity metric functions are based on the property-independent literal information extracted from each instance. The literal information $l = \{l_1, l_2, \ldots, l_n\}$ is similar to a virtual document generated from an instance. For an instance pair (a, b), a similarity metric function d_i maps the extracted literal information pair (l_i^a, l_i^b) to a real number in the range of $[0, 1]$.

2.2   Framework

The framework of our approach is shown in Figure 1. We extract literal information for each instance from the property-value descriptions. To get sufficient literal information for the similarity metrics, we conduct the following preprocessing for each property-value pair:

– A real-world object may be represented by an instance or a piece of text. For consistency, if the value is a URI which represents another instance, we will replace it by the label of that instance. Most of the instances have a label value which usually belongs to the property `rdfs:label` or some other common properties[10]. If the label of an instance is unavailable, we can replace it by its text description.
– We will also replace each property by its label. If the label is unavailable, we will use the last token (normalized) of the URI instead, e.g., "place of death" for `fb:place_of_death`.

For data sources A and B which contain $|A|$ and $|B|$ instances respectively, there are $|A| \times |B|$ possible instance pairs. This is unacceptable since both $|A|$ and $|B|$ can be one million or even larger. We use a simple pre-match method to sift the possible matching pairs. An inverted index is built for instances of some key words in their descriptions. The instances sharing the same keys in the index are considered to be candidate matching instances. Our preliminary experiments show that this sifting method can greatly reduce the number of pairs to test for a match, without losing much recall (the recall is over 0.9).

After preprocessing and literal information extraction, we compute the similarity vectors for the candidates. To train a binary classifier based on the similarity vectors, we need to label some of them into class -1 (non-matching) or class 1 (matching). Thus the problem of instance matching can be solved by classifying the unlabeled instance pairs. The existing matching instance pairs in LOD can be considered as labeled data which may help to train the classifier. We employ a transfer learning algorithm to improve the effect of the helping.

3 Feature Extraction

We extract some property-independent information from each instance and then compute the similarity vectors for instance pairs based on this information. Since the performance of a machine learning algorithm depends a lot on the quality of feature extraction, the work in this section is vital for the next step.

3.1 Literal Information Extraction

We extract several sets of literal information from each instance. The first is the text information set l_{label}, that is the label of an instance. The label is the human-readable name for an instance, such that it can help people to identify the real-world object. So labels are discriminative for instance matching. Next, we extract the remaining text information from the instance. These sets are divided into two parts. One is $l_{property}$ which consists of the text information from properties. The other is the text information from the values. The number of words in a value has a certain meaning. If the value only contains one word, this word can be a specific symbol such as the ISBN for a book. If the number of words is small, these words are likely to be the name of something, e.g. "Palo Alto". If there are a lot of words in the value, they may be a text description. These three kinds of values may play different roles in the problem of instance matching. So we extract them as l_{single}, l_{short} and l_{long} respectively.

Besides the large amount of text information, there are also other types of literal information in the instance descriptions. The common ones we used are dates, numbers and links. In contrast to the text information, these types of literal information are more useful for instance matching. If two instances share some dates, numbers or links, they are likely to match. So we additionally extract them as l_{date}, l_{number} and l_{link}. Note that:

Table 1. Overall Statistics on Extraction Results

Dimension Num	Metric Function	Combination of Literal Information
1	IdfSim	l_{single}
2	TopIdfSim	l_{single}
3	IdfSim	$l_{single} \cup l_{short} \cup l_{label}$
4	TopIdfSim	$l_{single} \cup l_{short} \cup l_{label}$
5	CosSim	$l_{single} \cup l_{short} \cup l_{label} \cup l_{property} \cup l_{long}$
6	IdfSim	$l_{single} \cup l_{short} \cup l_{label} \cup l_{property} \cup l_{long}$
7	TopIdfSim	$l_{single} \cup l_{short} \cup l_{label} \cup l_{property} \cup l_{long}$
8	EditSim	l_{label}
9	CountSim	l_{label}
10	CountSim	l_{date}
11	CountSim	l_{number}
12	CountSim	l_{link}

- There are many forms of dates. For convenience, we only extract the year part of each date and the other parts are treated as text.
- Some dates and numbers may be included in the texts. We use meticulous string processing to find them.

3.2 Similarity Metrics

Different similarity metric functions are used for different types of literal information. As shown in Table 1, a 12-dimensional similarity vector is generated for each instance pair.

For the text information, we use three functions, CosSim, IdfSim and TopIdfSim. CosSim is a common similarity metric for texts. It computes the $TF \cdot IDF$[7] weights for the words from two word sets and then computes their *cosine similarity*. Furthermore, in the particular problem of instance matching, the IDF weights are more important. Some common words or common words for the domain may appear frequently in the descriptions of many instances. These words with high TF weights but low IDF weights do not much help match instances. While if a word only appears once in each data source, the two instances that contain it are likely to match. According to this idea, IdfSim and TopIdfSim are designed based on the IDF weights of words. IdfSim is similar to CosSim which just removes the TF weights. For word sets T_1 and T_2, TopIdfSim computes the similarity of W_1 and W_2, where W_i is a subset of T_i which consists of the words with highest IDF weights in T_i. It is computed by:

$$\text{TopIdfSim}(T_1, T_2) = \frac{\sum_{w \in W_1 \cap T_2} \text{IDF}(w) + \sum_{w \in W_2 \cap T_1} \text{IDF}(w)}{\sum_{w \in W_1} \text{IDF}(w) + \sum_{w \in W_2} \text{IDF}(w)} \quad (1)$$

These three similarity metric functions act on the combinations of the extracted word sets of text information. The combining strategy is based on the relaxed

inclusion relation of these word sets from different instances, that is l_{single} may be included in l_{short} or l_{label}, and l_{long} main contains all the other word sets.

Among the sets of text information, l_{label} is different from the others in two ways as follows:

1. We only extract one label for each instance; so it can be treated as a set of words or a string.
2. Since each word in a label can be significant for recognizing the entity, to match two instances, the matching words of their labels are more important than the non-matching ones.

So we design another two similarity metrics for l_{label}, which are EditSim and CountSim.

$$\text{EditSim}(l_{label}^a, l_{label}^b) = 1 - \frac{\text{EditDistance}(S_a, S_b)}{\text{Max}(|S_a|, |S_b|)} \tag{2}$$

Where S_a stands for the string form of l_{label}^a and $\text{EditDistance}(S_a, S_b)$ is a classical string-distance measure, which represents the minimum number of editing operations needed to make S_a and S_b the same. Each operation can be deleting a character, inserting a character or changing a character in either S_a or S_b.

$$\text{CountSim}(l_{label}^a, l_{label}^b) = \frac{1 - 2^{-|\{w|w \in l_{label}^a \cap l_{label}^b\}|}}{1 - 2^{-\lceil(|l_{label}^a| + |l_{label}^b|)/2\rceil}} \tag{3}$$

The literal information sets of dates, numbers and links also have the second characteristic of the labels. So we use CountSim on them to generate similarities. Note that two numbers are considered to be the same one if their difference is lower than a threshold.

4   Learning Algorithms

After extracting the feature vectors, we can train the classifier for instance matching. There are many machine learning algorithms for the binary classification problem. We need to carefully choose the appropriate ones according to the characteristic of the input data.

4.1   Basic Learning Model

In our problem, the input data may contain millions instance pairs. So some methods with high time cost such as *Neural Networks* and *SVM* with a complex kernel are eliminated. After observing the feature space via a preliminary experiment, we found that the data of the two classes are not linearly separable. A typical example is shown in Figure 2. x and y are two dimensions of the similarity vector. The positive and negative regions represent the two classes. Figure 2 indicates that for a similarity vector, if x is greater than a threshold, it belongs to the class of matching instance pairs when y is large, but if x is less than the threshold, it belongs to the matching class when y is small. From the
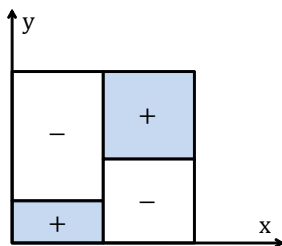
Fig. 2. A typical example of the feature space

perspective of each single dimension, this situation does not meet our intuition. The value of each dimension describes the similarity of two instances based on a certain metric. The higher the value is, the more likely they will match. But from the perspective of the correlation between the dimensions, such a situation is reasonable. We will give an example to explain it.

On one hand, given two instance pairs (p_1, p_2) and (q_1, q_2), their similarity vectors are u and v. We assume that $u_6 = v_6 = 0.3$, $u_5 = 0.1$ and $v_5 = 0.3$ where $u_i(v_i)$ stands for the ith dimension of the similarity vector $u(v)$. Probably, instance p_1 consists of long texts and instance p_2 consists of short texts or single words, so that $u_5 = 0.1$. While q_1 and q_2 both consist of long texts which share some common (for the domain) words, so $v_5 = 0.3$. Furthermore, p_1 and p_2 may share some important words such that $u_6 = 0.3$. While the value 0.3 of v_6 may be obtained by the common words. According to the inference above, (p_1, p_2) is likely to match while (q_1, q_2) is not. On the other hand, an instance pair with large values of both dimensions 5 and 6 is likely to match.

Since the feature space is so complex, some linear classifiers are inapplicable here, e.g. *linear regression* and *SVM*. Actually, the dimensions of our similarity vector have some implicit associations, especially the ones generated from the same combination of text sets. So the model we needed should properly handle such associations for classification.

Consider the *decision tree* model: A decision tree is a binary tree. Each non-leaf node t in the tree has a dimension-number k_t and a threshold σ_t and each leaf node has a label of one of the two classes. When a testing vector is sent to a non-leaf node t, it will be sequentially sent to a child node according to whether the value of k_tth dimension of the vector is greater than σ_t. So a testing vector will be sent to the root of the tree and finally get to a leaf node. The label of the leaf node will be the class the vector belongs to. The vectors which arrive at node t have passed the ancestors of t, i.e. A_t. In this way, the associations between dimension of k_t and the dimensions of $\{k_a | a \in A_t\}$ are taken into consideration for classification.

4.2   AdaBoost with Random Forest

AdaBoost[12] is the predecessor of the transfer learning algorithm we will use. It combines several weak classifiers to get a powerful classifier via an iterative process.

In each round of iteration, it employs a basic learning model to train a weak classifier with the current weighted examples. Then it increases the weights of the examples which are incorrectly classified by the weak classifier, so that these "difficult" ones will be classified better in the following rounds. In this way, *AdaBoost* may improve the performance of the basic learning model.

We found that some classical decision tree algorithms do not show good performance working with *AdaBoost*. In the training phase, the decision tree will continually grow to satisfy the training data. Since *AdaBoost* combines the examples from source and target domain for training, the trained tree-classifier will achieve good performance on both source and target domain examples. This leads to a situation where few incorrectly classified examples can be found during the iterations and the weights are hardly changed.

To solve this problem, we choose a specific decision tree model, *Random Forest*. A random forest contains several decision trees. Each of them is trained with a newly constructed training set, which is chosen by randomly picking some examples with replacement from the whole training data. The classification result of the random forest is a voting result of these trees. Due to the chosen strategy of the training set, most trees will only be good at classifying the examples in the dominating distribution. The remaining examples will be incorrectly classified so that they will be reweighted. Our preliminary experiments show that as a basic learning model of *AdaBoost*, *Random Tree* is superior to the other decision tree models, e.g. *J48Tree*[20].

4.3   Transfer Learning

To train an efficient classifier, a number of training examples are needed and should be labeled manually. To reduce the manual work, we want to utilize the existing matching instance pairs in LOD for help. But most machine learning methods work well only under the assumption that the training and testing data are drawn from the same feature space and the same distribution[24]. Training data which is generated from the existing matching instance pairs does not meet this requirement. *Transfer learning*[24] can utilize the data from different distributed domains (*source domain*) to help the target task, thus reducing the need for training data in the *target domain* (the domain of testing data).

There are two main kinds of transfer learning methods: the *instance-transfer* and the *feature-representation-transfer*. The former assumes that the feature spaces of the source and target domain are the same while the distributions of the data from the two domains are different. Such methods try to find examples from the source domain which can be reused to help the target domain task. The latter assumes that the feature spaces of the source and target domain are different. Such methods focus on finding the "good" common features which reduce the difference between the source and target domains and the error of classification.

The property matching independent similarity vectors we generated naturally compose a common feature space for all the data source pairs. So we employ *TrAdaBoost*[8] for help, which is a classical algorithm for *instance-transfer*

learning. *TrAdaBoost* is an extension of the *AdaBoost*[12] algorithm. It assumes that the feature spaces of source and target domain are the same while the distributions of the data from the two domains are different. Due to the different distributions, some of the source domain data may be helpful in training the classifier for the target domain but some of it may not be and could even be harmful. *TrAdaBoost* iteratively adjusts the weighting of the source domain data to reduce the effect of the "bad"source data. In each round of iteration, *TrAdaBoost* trains the basic classifier on the weighted source and target data. The source domain examples which are classified incorrectly by the classifier are considered to be the bad source data, so their weights are reduced. Meanwhile, *TrAdaBoost* uses the same strategy as *AdaBoost* that is to increase the weights of the incorrectly classified examples in the target domain.

Random Forest is a suitable basic learning model for *TrAdaBoost*, as well as for *AdaBoost*. The interface of *TrAdaBoost* is generic. We can directly apply it on the problem of instance matching by treating the instance pairs from a pair of data sources, which are to be matched, as the target domain, and the existing matching information between another pair of data sources as the source domain. But not all the source domains can help to train the classifier on the target domain via *TrAdaBoost*. A source domains can be harmful if its distribution is quite different from that of the target domain.

The problem of how to automatically choose a helpful source domain has not been theoretically solved yet[9]. Intuitively, the more similar the distributions of the two domains are, the more likely the source domain can help. In the problem of instance matching, the distribution of a domain is decided by the heterogeneity between the ways of describing objects which are used by the data sources (We call it describing heterogeneity for short). So when we want to match the instances of a data source pair (A, B), we should use another pair (C, D) for help, such that the describing heterogeneities of (A, B) and (C, D) are similar.

5   Experiments

First, we will show the experimental results of our proposed approach without transfer learning. We use the dataset provided by the data interlinking track of IM@OAEI2010 and compare our approach with the participants'. We chose this dataset because many others are not from LOD. Then we will give some comparative experiments to show whether a source domain we chose from LOD is helpful to instance matching via transfer learning.

5.1   Without Transfer Learning

The goal of the data interlinking track of IM@OAEI2010 is to find all the `owl:sameAs` links from four data sources to the ones in LOD. These four data sources are also in LOD; they are:

- Sider[7], about some drugs and their effects.
- DrugBank[8], about drugs and their chemical, pharmaceutical and pharmacological information.
- DiseaSome[9], about disorders and genes.
- DailyMed[10], about marketed drugs and chemical structure, mechanism of action, indication, usage, contraindications and adverse reactions for the drugs.

These data sources are already linked to LOD and the existing links will be treated as the standard answers. The well-known *Recall*, *Precision* and *F-Measure* are used for evaluation.

Two teams, ObjectCoref and RiMOM took part in this track. Their reports of results can be found in [32] and [16]. ObjectCoref[17] uses a self-learning framework to iteratively extend a kernel of matching instances. In each round of iteration, the most discriminative property-value pair is learned from the kernel for the further matching. RiMOM[19] is a multi-strategy ontology matching framework. It combines three strategies when facing an instance matching problem.

1. Edit distance between labels of two instances.
2. Cosine of the $TF \cdot IDF$ vectors for the text description of the instances to match.
3. Cosine of the $TF \cdot IDF$ vectors for the text description of the instances related to the ones to match.

In Table 2, we give the results of matching instances for each data source pair. RiMOM also gave their results on some other data source pairs which are not shown here, since we can not find the dumps of those data sources. For each testing data source pair, we randomly labeled 5% of the similarity vectors as training data (no more than 2000 for these datasets). Obviously, our proposed approach works better than ObjectCoref and RiMOM on these datasets.

For ObjectCoref, the process of learning discriminative property-value pairs depends on the lax matches of properties from different data sources. From the report of ObjectCoref[16], we can see that the matching properties found for these datasets are mainly about names and aliases. By analyzing the data, we find that Sider, DrugBank and DayliMed contain a lot of aliases for each instance, while DiseaSome does not. Furthermore, some non-matching instances have similar aliases. So ObjectCoref got high recall and low precision on Sider-DrugBank and Sider-DailyMed, but low recall and high precision on Sider-DiseaSome. In general, ObjectCoref did not get good performance since the properties of names and aliases do not match well. In contrast, RiMOM is a property matching independent approach. But the similarity metric that combines the three strategies is not accurate enough for instance matching.

---

[7] `http://sideeffects.embl.de/`

[8] `http://www.drugbank.ca/`

[9] `http://http://diseasome.kobic.re.kr/`

[10] `http://dailymed.nlm.nih.gov/`

Table 2. Compare with the Participants of IM@OAEI2010

Data Set	Approach	Recall	Precision	F-Measure
Sider-DrugBank	ObjectCoref	0.996	0.302	0.464
	RiMOM	0.342	0.961	0.504
	AdaBoost	0.859	0.952	**0.903**
Sider-DiseaSome	ObjectCoref	0.668	0.837	0.743
	RiMOM	0.315	0.837	0.458
	AdaBoost	0.726	0.875	**0.794**
Sider-DailyMed	ObjectCoref	0.999	0.548	0.708
	RiMOM	0.567	0.706	0.629
	AdaBoost	0.672	0.805	**0.733**
Sider-DBpedia	RiMOM	0.482	0.717	0.576
	AdaBoost	0.643	0.639	**0.641**
DailyMed-DBpedia	RiMOM	0.246	0.293	0.267
	AdaBoost	0.373	0.377	**0.375**

Table 3. Transfer GeoNames-DBpedia to LinkedGeoData-DBpedia

Training Examples	AdaBoost	AdaBoost(Source)	TrAdaboost
900	0.284	0.372	**0.378**
1500	0.383	0.396	**0.432**
3000	0.444	0.416	**0.458**
6000	**0.524**	0.450	0.516
15000	**0.544**	0.491	0.536

We noticed that our proposed approach has an enormous advantage on the data set Sider-DrugBank. The probable reason is that we can make use of the names and aliases for instance matching. Our approach eliminates the ill effects of the duplicate names by giving them low IDF weights.

5.2 With Transfer Learning

We choose GeoNames[11], LinkedGeoData[12] and DBpedia as the datasets for the experiments on transfer learning. GeoNames and LinkedGeoData are data sources in LOD. Both of them are about geographic information and have `owl:sameAs` links to DBpedia. GeoNames and LinkedGeoData have similar behaviors in describing real-world objects. So the the describing heterogeneities of GeoNames-DBpedia and LinkedGeoData-DBpedia are similar. We try to use the

---

[11] http://www.geonames.org/

[12] http://linkedgeodata.org/

information on the existing matching instances between GeoNames and DBpedia to help matching LinkedGeoData and DBpedia.

The result is shown in Table 3. **AdaBoost** denotes the *AdaBoost* model applied only on the training data from the target domain (LinkedGeoData-DBpedia). **AdaBoost(Source)** and **TrAdaBoost** respectively denote the *AdaBoost* and *TrAdaBoost* model applied on the training data from both domains. **Training Examples** denotes the number of training instance pairs we labeled in the target domain. 900 examples are about 0.01% of all the pre-matching instance pairs between LinkedGeoData and DBpedia. We can see that the source domain we chose is really helpful via *TrAdaBoost*. But directly using the source domain for training can be harmful. Furthermore, the less training data there is in the target domain, the more the source domain can help. If there is efficient training data in the target domain, the source domain is entirely useless. These experimental results match our intuition about transfer learning.

6   Related Work

Although the problem of instance matching has emerged along with the development of LOD in recent years, a similar problem, *Record linkage*, was examined much earlier. Thus a lot of relevant approaches have been proposed.

6.1   Record Linkage

Record linkage, also known as *duplicate detection* or *object identification*, is a classical problem in the area of databases. The goal of record linkage is to determine the pairs of records that are associated with the same entity across various data sets. It is similar to the instance matching problem. For more than five decades, the traditional database community has discussed this problem a lot. Some surveys can be found in [33], [34] and [11].

The early approaches focus on similarity metrics for single field (column) matching. Many of them are even widely used today, such as edit distance, *Q*-gram distance, etc. The similarity vector in our work is based on TF·IDF[7] and its variants.

The approaches for multi-field matching are mainly based on probabilistic models, developed by the machine learning community. The learning methods were applied on record linkage by extracting the feature vector of similarities from the comparable fields. Such approaches classify the record pairs as matching or not, employing CART[6], SVM[3] and so on. Semi-supervised, active and unsupervised learning algorithms have been proposed to reduce the demand for training data. Unlike the properties of linked data sources, the number of fields in a record linkage problem is usually quite small. Thus the comparable fields can be easily found manually. So the data base community doesn't pay attention to the problem known as *Structural heterogeneity*. A simple schema-independent method has also been proposed which treats the whole record as one large field. But experiments in [3] show that SVM based on the similarity

metric of multiple comparable fields usually outperforms it. It's easy to see that such an elaborate similarity metric benefits record linkage when the fields can well match. Some distance-based methods which have also been proposed for for multi-field matching do not employ machine learning algorithms[13][1], but the distance measures are also based on schema matching.

Some other work focuses on postprocessing. The main idea is that the matching relations between records or fields should be consistent[2][25]. Such methods based on graph model can play an important role in instance matching problems as the linked data settings are more structured. They are complementary to our proposed work which uses only the literal information.

6.2   Instance Matching

Some of the existing links in LOD are discovered manually with some tools. One well-known instance matching tool is the Silk Link Discovery Framework[31] which allows setting link specifications for given data sources. Domain related knowledge is required to design such specifications.

The automatic instance matching approaches are often domain specific or property matching dependent. The approach proposed in [29] matches the FOAF instances using SVM. Since all of the instances are described with FOAF, the features for classification are easily determined according to the limited number of properties. More domain-specific approaches can be found in [27] and [28]. Among the domain-independent approaches, [14] matches instances according to their inverse functional properties (IFP). Such properties are not sufficient in LOD, so [15] tries to find more IFPs with a statistical method. ObjectCoref[17] employs a self-learning framework to iteratively find the discriminative property-value pairs for instance matching, which are lax IFPs. RAVEN[22] applies active learning techniques for instance matching. Both ObjectCoref and RAVEN match the properties from different data sources by measuring value similarities. Similar ideas are proposed in the domain of *schema matching*[26].

Finally, some papers focus on improving the efficiency of instance matching. [21] limits the number of candidate instance pairs to match based on the triangle equation. [30], [23] and [18] generate candidates by indexing some key words of instances. This kind of method can be applied to optimize ours.

7   Conclusion and Feature Work

In this paper, we presented a property matching independent approach for instance matching. We transformed the problem of instance matching to a classification problem, by designing a novel feature vector of high-level similarity metrics. Suitable learning models were selected according to the feature space. Our experimental results on the datasets of IM@OAEI2010 shown that such a feature vector is reasonable for instance matching, and our approach performed much better than the contest participants. Furthermore, we tried to utilize the information of existing matches in LOD to help match new data sources via a

transfer learning algorithm. The experiments also show that such information is really helpful.

In future work, we will try to explore the following issues:

- The information on property matching and the relationships between instances can be taken into consideration in the similarity metrics. It may enrich the features for instance matching.
- *Random Forest* and *Adaboost* are similar in the result of classification[5]. Cooperation of *Random Forest* and *TrAdaboost* can be explored.
- The number of dimensions of the current similarity feature space is too low for machine learning with so many matching instances in LOD. More property matching independent similarity metrics need to be designed to make the best use of the information in the existing matches.
- We hope to find a powerful way to automatically choose a helpful source domain for the problem of instance matching.

References

1. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 586–597. VLDB Endowment (2002)
2. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Machine Learning 56(1), 89–113 (2004)
3. Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., Fienberg, S.: Adaptive name matching in information integration. IEEE Intelligent Systems 18(5), 16–23 (2003)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) 5(3), 1–22 (2009)
5. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
6. Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.: Efficient data reconciliation. Information Sciences 137(1), 1–15 (2001)
7. Cohen, W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. ACM SIGMOD Record 27, 201–212 (1998)
8. Dai, W., Yang, Q., Xue, G., Yu, Y.: Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning, pp. 193–200. ACM (2007)
9. Eaton, E., desJardins, M., et al.: Selective transfer between learning tasks using task-based boosting. In: Twenty-Fifth AAAI Conference on Artificial Intelligence (2011)
10. Ell, B., Vrandečić, D., Simperl, E.: Labels in the Web of Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 162–176. Springer, Heidelberg (2011)
11. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1), 1–16 (2007)
12. Freund, Y., Schapire, R.: A Desicion-Theoretic Generalization of On-line Learning and an Application to Boosting. In: Vitányi, P.M.B. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 23–37. Springer, Heidelberg (1995)
13. Guha, S., Koudas, N., Marathe, A., Srivastava, D.: Merging the results of approximate match operations. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, vol. 30, pp. 636–647. VLDB Endowment (2004)

14. Hogan, A., Harth, A., Decker, S.: Performing object consolidation on the semantic web data graph (2007)
15. Hogan, A., Polleres, A., Umbrich, J., Zimmermann, A.: Some entities are more equal than others: statistical methods to consolidate linked data. In: 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic, NeFoRS 2010 (2010)
16. Hu, W., Chen, J., Cheng, G., Qu, Y.: Objectcoref & falcon-ao: results for oaei 2010. In: Ontology Matching, p. 158 (2010)
17. Hu, W., Chen, J., Qu, Y.: A self-training approach for resolving object coreference on the semantic web. In: Proceedings of the 20th International Conference on World Wide Web, pp. 87–96. ACM (2011)
18. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall (2011)
19. Li, J., Tang, J., Li, Y., Luo, Q.: Rimom: A dynamic multistrategy ontology alignment framework. IEEE Transactions on Knowledge and Data Engineering 21(8), 1218–1232 (2009)
20. Loh, W.: Classification and regression tree methods. In: Encyclopedia of Statistics in Quality and Reliability (2008)
21. Ngomo, A., Auer, S.: Limes-a time-efficient approach for large-scale link discovery on the web of data. In: Proceedings of IJCAI (2011)
22. Ngomo, A., Lehmann, J., Auer, S., Höffner, K.: Raven–active learning of link specifications. In: Ontology Matching, p. 25 (2011)
23. Niu, X., Rong, S., Zhang, Y., Wang, H.: Zhishi.links results for oaei 2011. In: Ontology Matching, p. 220 (2011)
24. Pan, S., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering 22(10), 1345–1359 (2010)
25. Pasula, H., Marthi, B., Milch, B., Russell, S., Shpitser, I.: Identity uncertainty and citation matching. In: Proceedings of NIPS 2002 (2002)
26. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. The VLDB Journal 10(4), 334–350 (2001)
27. Raimond, Y., Sutton, C., Sandler, M.: Automatic interlinking of music datasets on the semantic web. In: Proceedings of the 1st Workshop about Linked Data on the Web. Citeseer (2008)
28. Sleeman, J., Finin, T.: Computing foaf co-reference relations with rules and machine learning. In: Proceedings of the Third International Workshop on Social Data on the Web (2010)
29. Sleeman, J., Finin, T.: A machine learning approach to linking foaf instances. In: Spring Symposium on Linked Data Meets AI. AAAI (January 2010)
30. Song, D., Heflin, J.: Automatically Generating Data Linkages Using a Domain-Independent Candidate Selection Approach. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 649–664. Springer, Heidelberg (2011)
31. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 650–665. Springer, Heidelberg (2009)
32. Wang, Z., Zhang, X., Hou, L., Zhao, Y., Li, J., Qi, Y., Tang, J.: Rimom results for oaei 2010. In: Ontology Matching, p. 195 (2010)
33. Winkler, W.: The state of record linkage and current research problems. In: Statistical Research Division, US Census Bureau. Citeseer (1999)
34. Winkler, W.: Overview of record linkage and current research directions. In: Bureau of the Census. Citeseer (2006)