Instance-Based Ontological Knowledge Acquisition

Lihua Zhao^{1,2} and Ryutaro Ichise^{2,1}

^{1} The Graduate University for Advanced Studies
^{2} National Institute of Informatics, Tokyo, Japan
{lihua, ichise}@nii.ac.jp

Abstract. The Linked Open Data (LOD) cloud contains tremendous amounts of interlinked instances, from where we can retrieve abundant knowledge. However, because of the heterogeneous and big ontologies, it is time consuming to learn all the ontologies manually and it is difficult to observe which properties are important for describing instances of a specific class. In order to construct an ontology that can help users easily access to various data sets, we propose a semi-automatic ontology integration framework that can reduce the heterogeneity of ontologies and retrieve frequently used core properties for each class. The framework consists of three main components: graph-based ontology integration, machine-learning-based ontology schema extraction, and an ontology merger. By analyzing the instances of the linked data sets, this framework acquires ontological knowledge and constructs a high-quality integrated ontology, which is easily understandable and effective in knowledge acquisition from various data sets using simple SPARQL queries.

Keywords: Semantic Web, linked data, ontology integration, knowledge acquisition, machine learning.

1 Introduction

The Linked Open Data (LOD) is a collection of machine-readable structured data with over 31 billion Resource Description Framework (RDF) triples interlinked by around 504 million *SameAs* links (as of Sep. 2011). Instances are represented using the Uniform Resource Identifier (URI), and identical instances are linked with the built-in OWL property *owl:sameAs* [3]. The Web Ontology Language (OWL) is a semantic markup language developed as a vocabulary extension of the RDF with more vocabularies for describing properties and classes [2]. RDF Schema is a simple vocabulary for describing properties and classes of RDF resources. The OWL 2 Web Ontology Language [16] provides classes and properties as the old OWL 1[2], but with richer data types, data ranges, and disjoint properties, etc.

The LOD cloud has been growing rapidly over the past years and many Semantic Web applications have been developed by accessing the linked data sets [4]. However, in order to use the data sets, we have to understand their heterogeneous ontologies in advance. One possible solution for the ontology heterogeneity

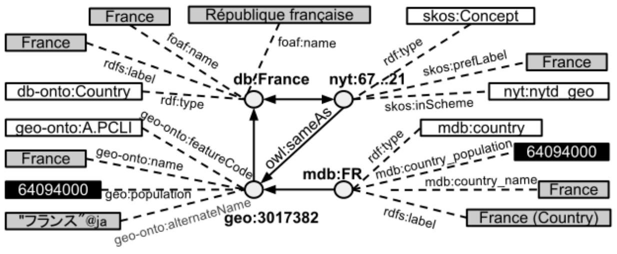

Fig. 1. Interlinked Instances of "France".

problem is constructing a global ontology that integrates various ontologies in the LOD cloud. Ontologies commonly consist of classes, properties, and relations between them. Although the built-in properties owl:equivalentClass and owl:equivalentProperty are designed to link classes or properties with the same concept, there are only few links at a class or property level [7]. Hence, it is difficult to directly retrieve equivalent classes and properties for integrating ontologies of various data sets.

In order to integrate ontologies of various data sets, we need to identify related classes and properties of the ontologies. Since the same instances are linked by *owl:sameAs*, we can create undirected graphs with the linked instances. Fig. 1 shows the interlinked instances of "France" and each instance is described using properties and objects. As shown in Fig. 1, all the properties (labeled on the dotted line) connected with the grey boxes (objects) represent the name of "France" and the properties connected to the black boxes represent the population. By analyzing the graph patterns, we can observe how the same classes and properties are represented differently in various data sets and integrate them.

Other than integrating related classes and properties, we also need frequently used core classes and properties to construct a high-quality integrated ontology. Machine learning methods such as association rule learning and rule-based classification can be applied to discover core properties for describing instances in a specific class. Apriori is a well-known algorithm for learning association rules in a big database [1], while the rule-based learning method - Decision Table can retrieve a subset of properties that leads to high prediction accuracy with cross-validation [10].

In this paper, we propose a framework that semi-automatically integrates heterogeneous ontologies for the linked data sets. The integrated ontology consists of frequently used core classes and properties that can help Semantic Web application developers easily understand the ontology schema of the data sets. Furthermore, the integrated ontology also includes related classes and properties, with which we can integrate data sets and find missing links between instances.

This paper is organized as follows. In Section 2, we discuss some related work and the limitation of their methods. In Section 3, we introduce our semi-automatic ontology integration framework in details. The experiments are discussed in Section 4. We conclude and propose future work in Section 5.

2 Related Work

The authors in [11] introduced a closed frequent graph mining algorithm to extract frequent graph patterns from the Linked Data Cloud. Then, they extracted features from the entities of the graph patterns to detect hidden *owl:sameAs* links or relations in geographic data sets. They applied a supervised learning method on the frequent graph patterns to find useful attributes that link instances. However, their approach only focused on geographic data and did not discuss about what kind of features are important for finding the hidden links.

A debugging method for mapping lightweight ontologies is introduced in [13]. They applied machine learning method to determine the disjointness of any pair of classes, with the features of the taxonomic overlap, semantic distance, object properties, label similarity, and WordNet similarity. Although their method performs better than other ontology matching systems, their method is limited to the expressive lightweight ontologies.

In [14], the authors focused on finding concept coverings between two sources by exploring disjunctions of restriction classes. Their approach produces coverings where concepts at different levels in the ontologies can be mapped even there is no direct equivalence. However, the work is mainly for specific domains and the alignments of ontologies are limited between two resources.

In contrast to the research described above, our approach retrieves related ontology schema and frequently used core properties and classes in each data set. Our method is domain-independent and successfully integrates heterogeneous ontologies by extracting related properties and classes that are critical for interlinking instances. In addition, for the instances of specific classes, we can recommend core properties that are frequently used for the instance description.

3 Semi-automatic Ontology Integration Framework

Constructing a global ontology by integrating heterogeneous ontologies of the linked data can help effectively integrate various data resources. In order to create an integrated ontology and decrease the ontology heterogeneity problem, we focus on retrieving related classes and properties, top-level classes, and frequent core properties. We can extract related classes and properties from the interlinked instances using the graph-based ontology integration component. In addition, we also need the top-level classes and frequent core properties in each data set, which can be extracted using machine learning methods. For instance, the Decision Table algorithm can retrieve a subset of properties that leads to high prediction accuracy with cross-validation and the Apriori algorithm can discover properties that occur frequently in the instances of top-level classes.

In this paper, we propose a semi-automatic ontology integration framework, which is an extension of the previous work in [18]. The semi-automatic ontology integration framework is shown in Fig. 2, which consists of three main components: graph-based ontology integration, machine-learning-based ontology schema extraction, and an ontology merger. In the following, we will describe each component in details.

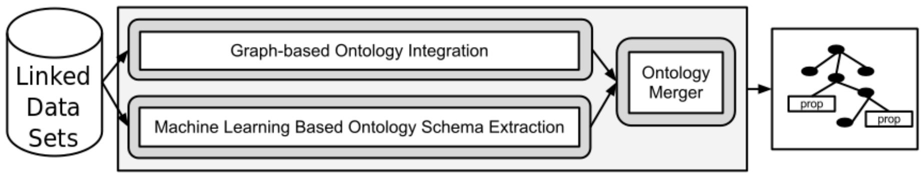
Linked Data Sets
Graph-based Ontology Integration
Machine Learning Based Ontology Schema Extraction
Ontology Merger
prop
prop

Fig. 2. Framework for Semi-automatic Ontology Integration.

3.1 Graph-Based Ontology Integration.

The graph-based ontology integration component semi-automatically finds related classes and properties by analyzing *SameAs* graph patterns in the linked data sets [18]. We will briefly describe this component, which is shown in Fig. 3. This component consists of five processes: graph pattern extraction, <Predicate, Object> collection, related classes and properties grouping, aggregation for all graph patterns, and manual revision.

Graph Pattern Extraction. Since the instances which refer to the same thing are interlinked by *owl:sameAs* in the LOD cloud, we collect all the linked instances and construct graph patterns according to the *SameAs* graphs extraction algorithm introduced in [18]. All the same *SameAs* graphs consist of a graph pattern, from which we can detect related classes and properties.

<Predicate, Object> Collection. An instance is described by a collection of RDF triples in the form of <subject, predicate, object>. Since a *SameAs* graph contains linked instances, we collect all the <Predicate, Object> (*PO*) pairs of the interlinked instances as the content of a *SameAs* graph and classify the *PO* pairs into five different types: Class, Date, URI, Number, and String.

Related Classes and Properties Grouping. We track subsumption relations to group related classes and apply different similarity matching methods to group related properties. In the following, we discuss how to retrieve and group related classes and properties from different types of *PO* pairs.

Class. For the PO pairs of type Class, we retrieve related classes from the most specific classes of the linked instances by tracking the subsumption relations such as owl:subClassOf and skos:inScheme. The classes and subsumption relations compose a tree, where the most specific classes are called leaf nodes in the tree.

Date and URI. We perform exact matching on the types of Date and URI, because even a slight difference of object values may refer to totally different properties.

Number and String. For the types of Number and String, the object values may vary in different data sets. For instance, the population of a country may

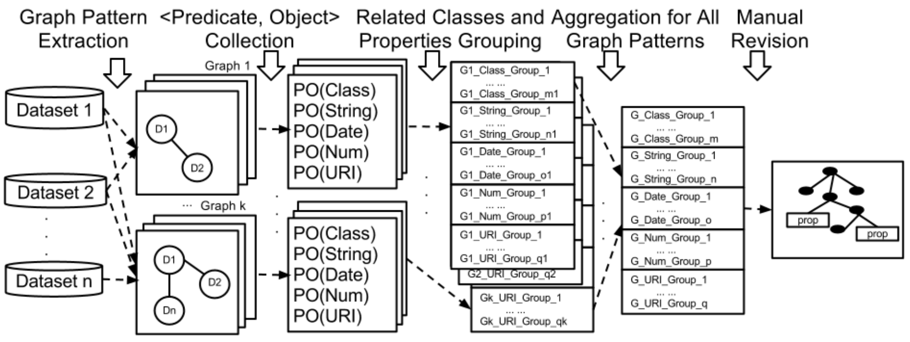

Fig. 3. Graph-Based Ontology Integration.

be slightly different in diverse data sets and the values in String may have different representations for the same meaning. Hence, in order to discover similar properties for the types of Number and String, we apply similarity matching approach by extending the methods introduced in [19].

The string-based and knowledge-based similarity matching methods are commonly used to match ontologies at the concept level [5]. In our approach, we adopted three string-based similarity measures, namely, JaroWinkler distance [17], Levenshtein distance, and n-gram, as introduced in [8]. String-based similarity measures are applied to compare the objects of *PO* pairs that are classified in String.

The knowledge-based similarity measures are applied to compare the preprocessed terms of predicates, because most of the terms have semantic meanings that can be recognized as a concept. To extract the concepts of predicate terms, we pre-process the predicates of *PO* pairs by performing natural language processing (NLP). We adopted nine knowledge-based similarity measures [15], namely, LCH, RES, HSO, JCN, LESK, PATH,WUP, LIN, and VECTOR, which are based on WordNet [6] (a large lexical database of English).

Aggregation for All Graph Patterns. In this step, we aggregate the integrated groups of classes and properties from all the graph patterns. An integrated ontology is automatically constructed with the integrated sets of related classes and properties, automatically selected terms, and the designed relations that link groups of related classes and properties to the integrated ontology schema.

Manual Revision. The automatically integrated ontology of this framework includes related classes and properties from different data sets. However, not all the terms of classes and properties are properly selected. Hence, we need experts to work on revising the integrated ontology by choosing a proper term for each group of properties, and by amending wrong groups of classes and properties. Since the integrated ontology is much smaller than the original ontology schema, it is a lightweight work.

The graph-based ontology integration component can discover related classes and properties from various data sets. By analyzing the extracted graph patterns, we detect related classes and properties which are classified into different data types: Date, URI, Number, and String. Similar classes are integrated by tracking subsumption relations and different similarity matching methods are applied on different types of PO pairs to retrieve similar properties. We automatically integrate related classes and properties for each graph pattern, and then aggregate all of them.

3.2 Machine-Learning-Based Ontology Schema Extraction

Although, the graph-based ontology integration method can retrieve related classes and properties from different ontologies, it misses some classes and frequent core properties. Therefore, we need another method to find top-level classes and frequent core properties, which are essential for describing instances.

By applying machine learning methods, we can find frequent core properties that are used to describe instances of a specific class. The Decision Table algorithm is a rule-based algorithm that can retrieve a subset of core properties and the Apriori algorithm can find a set of associated properties that are frequently used for describing instances. Hence, we apply the Decision Table and the Apriori algorithm to retrieve top-level classes and frequent core properties from the linked data sets.

In order to perform the machine learning methods, we randomly select a fixed number of instances for each top-level class from the data sets. For the data sets built based on ontology schema, we track subsumption relations to retrieve the top-level classes. For instance, we track owl:subClassOf subsumption relation to retrieve the top-level classes in DBpedia and track skos:inScheme in Geonames. However, some data sets use categories without any structured ontology schema. For this kind of data sets, we use the categories as top-level classes. For example, NYTimes instances are only categorized into people, locations, organizations, and descriptors. We use this strategy to collect the top-level classes in each data set, and then extract properties that appear more than the frequency threshold θ. The selected instances, properties, and top-level classes are used for performing machine learning methods.

Decision Table. The Decision Table is a simple rule-based supervised learning algorithm, which leads to high performance with simple hypothesis [10]. The Decision Table algorithm can retrieve a subset of core properties that can predict unlabeled instances with a high accuracy. Hence, properties retrieved by the Decision Table play an important role in the data description.

We convert the instances of linked data sets into data that is adaptable to the Decision Table algorithm. The data consists of a list of weights of properties and class labels, where the weight represents the importance of a property in an instance and the labels are top-level classes. The weight of a property in an instance is calculated in a similar way as the TF-IDF (Term Frequency - Inverse

Document Frequency), which is often used as a weighting factor in information retrieval and text mining [12]. The TF-IDF value reflects how important a word is to a document in a collection or corpus. The weight of each property in an instance is defined as the product of property frequency (PF) and the inverse instance frequency (IIF) in a similar way as the TF-IDF. The $pf(prop, inst)$ is the frequency of the property *prop* in the instance *inst*.

The inverse instance frequency of the property *prop* in the data set D is $iif(prop,\ D)$, calculated as follows:

$$iif(prop, D) = \log \frac{|D|}{|inst_{prop}|}$$

where $inst_{prop}$ indicates an instance that contains the property *prop*. The value of $iif(prop,\ D)$ is the logarithm of the ratio between the number of instances in D and the number of instances that contain the *prop*. If *prop* appears in *inst*, the weight of *prop* is calculated according to the following equation:

$$weight(prop, inst) = pf(prop, inst) \times iif(prop, D)$$

The properties retrieved with the Decision Table in each data set are critical for describing instances in the data set. Hence, we use these retrieved properties and top-level classes as parts of the final integrated ontology.

Apriori. Association rule learning method can extract a set of properties that occur frequently in instances. Apriori is a classic association rule mining algorithm, which is designed to operate on the databases of transactions. A frequent itemset is an itemset whose support is greater than the user-specified minimum support. Each instance in a specific class represents a transaction, and the properties that describe the instance are treated as items. Hence, the frequent itemsets represent frequently used properties for describing instances of a specific class. The frequent core properties can be recommended to the data publishers or help them find missing important descriptions of instances.

For each instance, we collect a top-level class and all the properties that appear in the instance as a transaction data. The Apriori algorithm can extract associated sets of properties that occur frequently in the instances of a top-level class. Hence, the retrieved sets of properties are essential for describing instances of a specific class. Furthermore, we can either identify commonly used properties in each data set or unique properties used in the instances of each class. Therefore, the properties extracted with the Apriori algorithm are necessary for the integrated ontology.

3.3 Ontology Merger

The third component is an ontology merger, which merges the ontology classes and properties extracted from the previous two components. The graph-based ontology integration component outputs groups of related classes and properties. On the other hand, the machine-learning-based ontology schema extraction component outputs a set of core properties retrieved by the Decision Table and a set of properties along with a top-level class retrieved using the Apriori.

We adopt OWL 2 for constructing an integrated ontology. During the merging process, we also add relations between classes and properties so that we can easily identify what kind of properties are used to describe instances of a specific class. We obey the following rules to construct the integrated ontology, where "ex-onto" and "ex-prop" are the prefixes of the integrated ontology.

Class. Related classes are collected from the graph-based ontology integration component and the top-level classes in each data set are collected from the machine-learning-based ontology schema extraction component.

Groups of classes from graph-based ontology integration. Related classes from different data sets are extracted by analyzing *SameAs* graph patterns and grouped into $cgroup_1, cgroup_2, ..., cgroup_z$. For each group, we automatically define a term *ex-onto:ClassTerm* for each group, where the *ClassTerm* is the most frequent term in the group. For each class $c_i \in cgroup_k$, we add a triple $<$ *ex-onto:ClassTerm*$_k$, *ex-prop:hasMemberClasses*, $c_i >$.

Classes from machine-learning-based ontology schema extraction. Top-level classes in each data set are added to the integrated ontology. If a top-level class $c_i \notin cgroup_k (1 \leq k \leq z)$, we create a new group $cgroup_{z+1}$ for c_i and a new term *ex-onto:ClassTerm*$_{z+1}$ for the new group. Then we add a triple $<$ *ex-onto:ClassTerm*$_{z+1}$, *ex-prop:hasMemberClasses*, $c_i >$.

Property. The extracted properties from two components are merged according to the following rules. At first, we extract the existing property type and domain information of each property from the data sets. The property type is mainly defined as rdf:Property, owl:DataTypeProperty, and owl:ObjectProperty. If the type is not clearly defined, we set the type as rdf:Property.

Groups of properties from graph-based ontology integration. Related properties from various data sets are extracted by analyzing *SameAs* graph patterns and grouped into $pgroup_1, pgroup_2, ..., pgroup_p$. For each group, we choose the most frequent term *ex-onto:propTerm*. Then, for each property $prop_i \in pgroup_t$ $(1 \leq t \leq p)$, we add a triple $<$ *ex-onto:propTerm*$_t$, *ex-prop:hasMemberProperties*, $prop_i >$.

Properties from machine-learning-based ontology schema extraction. We automatically add domain information for the properties retrieved using the Apriori method. For each property *prop* extracted from the instances of class c, we add a triple $<$ *prop*, *rdfs:domain*, $c >$, if it's not defined in the data set.

The ontology merger constructs an integrated ontology using the triples created as above. The global integrated ontology constructed with the ontology merger can help us easily access to various data sets and discover missing links. Furthermore, the domain information of the properties are automatically added using the results of the Apriori algorithm.

Table 1. Data Sets for Experiments.

Data Set	Instances	Selected Instances	Class	Top-level Class	Property	Selected Property
DBpedia	3,708,696	64,460	241	28	1385	840
Geonames	7,480,462	45,000	428	9	31	21
NYTimes	10,441	10,441	5	4	8	7
LinkedMDB	694,400	50,000	53	10	107	60

4 Experiments

In this section, we introduce the experimental data sets. Then we discuss whether we successfully retrieved related classes and properties using the graph-based ontology integration. We also discuss experimental results with the Decision Table and the Apriori algorithm that retrieve top-level classes and frequent core properties. Comparison with the previous work introduced in [18] and other ontology matching tools is also discussed in this section. At last, we discuss use cases with the integrated ontology and propose possible applications.

4.1 Data Sets

We selected DBpedia (v3.6), Geonames (v2.2.1), NYTimes and LinkedMDB from the LOD cloud to evaluate our framework. DBpedia is a cross-domain data set with about 8.9 million URIs and Geonames is a geographic domain data set with more than 7 million distinct URIs. NYTimes and LinkedMDB are both from media-domain with 10,467 and 0.5 million URIs, respectively.

The number of instances in our database are listed in the second column of Table 1. The graph-based ontology integration component uses all the instances in the data sets. For the machine learning methods, we randomly choose samples of the data sets to speed up the modeling process as well as to concern unbiased data size for each top-level class. We randomly selected 5000 instances per top-level class in Geonames and LinkedMDB, 3000 instances per top-level class in DBpedia, and used all the instances in NYTimes. The number of selected instances of DBpedia is less than 84,000, because some classes include less than 3000 instances.

The original number of classes and properties, the number of top-level classes and selected properties for the machine learning methods are listed in the Table 1. We track the subsumption relations such as owl:subClassOf and skos:inScheme to collect the top-level classes. Since there are a big number of properties in the data sets, we filter out infrequent properties that appear less than the frequency threshold θ. For each data set, we manually set a different frequency threshold θ as $\sqrt{n}$, where n is the total number of instances in the data set.

Table 2. Results for the Decision Table Algorithm.

Data Set	Average Precision	Average Recall	Average F-Measure	Selected Properties
DBpedia	0.892	0.821	0.837	53
Geonames	0.472	0.4	0.324	10
NYTimes	0.795	0.792	0.785	5
LinkedMDB	1	1	1	11

4.2 Graph-Based Ontology Integration

The graph-based ontology integration component uses all the interlinked instances in the data sets. With this component, we retrieved 13 different graph patterns from the *SameAs* Graphs [18]. In total, we extracted 97 classes from the data sets and grouped them into 48 new classes. Each group contains at least two classes and one class can belong to several groups. For instance, the schema in NYTimes is too general, so that the nyt:nytd_geo belongs to any group that has geographical information. Here, we give an example of the integrated class ex-onto:Country, which contains geo-onto:A.PCLI, geo-onto:A.PCLD, mdb:country, db-onto:Country, and nyt:nytd_geo. This group contains the classes about a country from Geonames, LinkedMDB, and DBpedia, except the general geographic class nyt:nytd_geo from NYTimes.

We retrieved 357 properties from the graph patterns using exact or similarity matching, which are integrated into 38 groups. Because of the heterogeneous infobox properties in DBpedia, some groups contain more than one DBpedia property. For instance, the properties geo-onto:population, mdb:country_population, db-onto:populationTotal, db-prop:populationTotal, and other eight DBpedia properties are integrated into the property ex-prop:population.

The graph-based ontology integration can retrieve related classes and properties from directly or indirectly linked instances by analyzing graph patterns.

4.3 Decision Table

The Decision Table algorithm is used to discover a subset of features that can achieve high prediction accuracy with cross-validation. Hence, we apply the Decision Table to retrieve core properties that are essential in describing instances of the data sets. For each data set, we perform the Decision Table algorithm to retrieve core properties by analyzing randomly selected instances of the top-level classes.

In Table 2, we listed the percentage of the weighted average of precision, recall, and F-measure. Precision is the ratio of correct results to all the results retrieved, and recall is the percentage of the retrieved relevant results to all the relevant results. The F-measure is a measure of a test's accuracy, that considers both the precision and the recall. The F-measure is the weighted harmonic mean of precision and recall, calculated as:

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

The F-measure reaches its best value at 1 and worst value at 0. A higher F-measure value means the retrieved subset of properties can well classify instances with unique and core properties. A lower F-measure fails to classify some instances, because the retrieved properties are commonly used in every instance. In the following, we will discuss the experimental results in each data set using the Decision Table algorithm.

DBpedia. The Decision Table algorithm extracted 53 DBpedia properties from 840 selected properties. For example, the properties db-prop:city, db-prop:debut, db-onto:formationYear, and db-prop:stateName are extracted from DBpedia instances. The precision, recall, and F-measure on DBpedia are 0.892, 0.821, and 0.837, respectively.

Geonames. We retrieved 10 properties from 21 selected properties, such as geo-onto:alternateName, geo-onto:countryCode, and wgs84_post:alt, etc. Since all the instances of Geonames are from geographic domain, the Decision Table algorithm can not well distinguish different classes with these commonly used properties. Hence, the evaluation results on Geonames are very low with 0.472 precision, 0.4 recall, and 0.324 F-measure.

NYTimes. Among 7 properties used in the data set, 5 properties are retrieved using the Decision Table algorithm. The extracted properties are skos:scopeNote, nyt:latest_use, nyt:topicPage, skos:definition, and wgs84_pos:long. In NYTimes, there are only few properties for describing news articles and most of them are commonly used in every instance. The cross-validation test with NYTimes are 0.795 precision, 0.792 recall and 0.785 F-measure.

LinkedMDB. The algorithm can classify all the instances in the LinkedMDB correctly with the 11 properties selected from 60 properties. Other than the commonly used properties such as foaf:page, rfs:label, we also extracted some unique properties such as mdb:performance_performanceid, mdb:writer_writerid, and director_directorid, etc.

The Decision Table algorithm retrieves a subset of core properties that are important to distinguish instances. We feed the extracted properties to the integrated ontology. However, the Decision Table can not find all the core properties in each data set.

4.4 Apriori

The Apriori algorithm is a classic algorithm for retrieving frequent itemsets based on the transaction data. For the experiment, we use the parameters upper and lower bound of minimum support as 1 and 0.2, respectively. We use the default minimum metric as 0.9 for the confidence metric. With a lower minimum support, we can retrieve more properties that frequently appear in the data.

Table 3. Examples of Retrieved Properties with the Apriori Algorithm.

Data Set	Class	Properties
DBpedia	db:Event	db-onto:place, db-prop:date, db-onto:related/geo.
	db:Species	db-onto:kingdom, db-onto:class, db-onto:family.
	db:Person	foaf:givenName, foaf:surname, db-onto:birthDate.
Geonames	geo:P	geo-onto:alternateName, geo-onto:countryCode.
	geo:R	wgs84_pos:alt, geo-onto:name, geo-onto:countryCode.
NYTimes	nyt:nytd_geo	wgs84_pos:long.
	nyt:nytd_des	skos:scopeNote.
LinkedMDB	mdb:actor	mdb:performance, mdb:actor_name, mdb:actor_netflix_id.
	mdb:film	mdb:director, mdb:performane, mdb:actor, dc:date.

We retrieved frequently appeared core properties using the Apriori algorithm. Some examples are listed in Table 3. The first column lists the experimental data sets, and the second column lists samples of the top-level classes in each data set. The third column lists some of the retrieved interesting or unique properties from each top-level class. As we can see from Table 3, the place, date and geographic properties are important for describing events. The best-known taxonomies such as kingdom, class, and family are also extracted by analyzing the data of species. From the LinkedMDB, we extracted mdb:actor_netflix_id, mdb:actor_name, and mdb:performance, that are critical for distinguishing different instances. Furthermore, the properties of director, performance, actor and date of a film are extracted from instances in the class mdb:film.

In DBpedia and LinkedMDB, we retrieved some unique properties in each class. However, for Geonames and NYTimes, we only retrieved commonly used properties in the data sets. From the instances of Geonames, we found commonly used properties such as geo-onto:alternateName, wgs84_pos:alt, and geo-onto:countryCode, etc. NYTimes only has few properties that are commonly used in every instance, except the wgs84_pos:long in the nyt:nytd_geo class and skos:scopeNote in the nyt:nytd_des class. Hence, the weighted average F-measure of nyt:nytd_geo and nyt:nytd_des are much higher than other classes.

We retrieved frequent sets of properties in most of the cases except in the db:Planet class. Because db:Planet contains 201 different properties for describing instances, which are sparsely used. In addition, we only retrieved db-onto:title and rdfs:type from db:PersonFunction and only rdfs:type property from db:Sales. This is caused by the lack of descriptions in the instances: most of the instances in db:PersonFunction and db:Sales only defined the class information without other detailed descriptions.

The set of properties retrieved from each class imples that the properties are frequently used for instance description of the class. Hence, for each property *prop* retrieved from the instances of class *c*, we automatically added <*prop*, rdfs:domain, *c*> to assert that *prop* can be used for describing instances in the class *c*. Therefore, we can automatically recommend missing core properties for an instance based on its top-level class.

Table 4. Extracted Classes and Properties

	Previous Work	Machine Learning		Current Work
	Graph-Based Integration	Decision Table	Apriori	Integrated Ontology
Class	97	50 (38 new)	50 (38 new)	135 (38 new)
Property	357	79 (49 new)	119(80 new)	453 (96 new)

4.5 Comparison

The graph-based ontology integration framework introduced in [18] only focuses on the related classes and properties acquisition, that may miss some core properties and classes. Hence, we applied machine learning methods to find out core properties for describing instances. The second column of Table 4 lists the number of classes and properties retrieved with the previous work - graph-based integration method. The next two columns list the number of classes and properties retrieved with the machine learning methods - Decision Table and Apriori. The last column is the final integrated ontology which merged the acquired ontological knowledge from two functional components: graph-based ontology integration and machine-learning-based ontology schema extraction, which includes Decision Table and Apriori algorithms.

With the graph-based ontology integration framework, we retrieved 97 classes and 357 properties, which are grouped into 49 and 38 groups, respectively. The final integrated ontology contains 135 classes and 453 properties that are grouped into 87 and 97 groups, respectively. Both of the Decision Table and the Apriori algorithms are performed on 50 selected top-level classes, among them 38 are not retrieved in the graph-based ontology integration. With the Decision Table, we extracted 79 properties, where 49 are not found in the graph-based ontology integration. The Apriori algorithm discovered 119 properties in total, where 80 properties are newly added. Based on the same data sets, Apriori can retrieve more properties than the Decision Table algorithm. Among the newly retrieved properties, 33 properties are retrieved from both Decision Table and Apriori.

By adding machine-learning-based ontology schema extraction component to the graph-based ontology integration, the final integrated ontology become more concrete with groups of related classes and properties, top-level classes, and core properties that are frequently used in instances. For each retrieved property, we automatically added property type definition and for the properties retrieved with the Apriori results, we automatically added domain information to indicate the relations between properties and classes.

Since most of the ontology matching tools fail to find alignments for the datasets that do not have a well designed ontology schema [9], we cannot use them to find alignments among DBpedia, Geonames, NYTimes, and LinkedMDB. The failure in the ontology alignment is caused by some ontologies that have ambiguous meaning of the concepts or the absence of corresponding concepts in the target dataset. However, our approach can find alignments for the poorly structured datasets by analyzing the contents of the interlinked instances.

4.6 Case Studies

In this section, we introduce some use cases with our integrated ontology. The class of db:Actor and mdb:actor are integrated into ex-onto:Actor, which can be used for discovering missing class information of the linked instances of actors. For instance, the db:Shingo_Katori is only described as a musical artist, but in fact he is also an actor and the DBpedia instance has a link to the mdb-actor:27092. Hence, we should add the class db-onto:Actor to the instance db:Shingo_Katori, because all the instances linked with mdb-actor should be an actor unless it is a wrong linkage.

If we want to link a person from different data sets, we can combine the class which indicates a person with some properties such as the birth date, the place of birth, and the name, etc. However, there exist various properties to describe the same kind of property. For example, we integrated 7 different properties that indicate the birthday of a person into the ex-prop:birthDate. Among them, only the property "db-onto:birthDate" has the default domain definition as db-onto:Person and has the highest frequency of usage, that appeared in 287,327 DBpedia instances. From the definitions of the properties and the number of instances which contain the corresponding properties, we can assume that the properties except "db-onto:birthDate" are mistakenly used when the data providers publish the DBpedia data. Therefore, we can suggest "db-onto:birthDate" as the standard property to represent the birthday of a person, and correct the other properties with this standard property.

Other than recommending standard properties, we also successfully integrated different property descriptions from diverse data sets. For instance, properties geo-onto:population, mdb:country_population, db-onto:populationTotal and other nine DBpedia properties are integrated into the property ex-prop:population. By combining the ex-onto:Country and ex-prop:population, we can detect the same country or countries with similar population.

5 Conclusion and Future Work

We proposed a semi-automatic ontology integration framework that can integrate heterogeneous ontologies by analyzing graph patterns of the interlinked instances and by applying machine learning methods. Grouping related classes and properties can reduce the heterogeneity of ontologies in the LOD cloud. The integrated ontology consists of related classes and properties, top-level classes and frequent core properties that can help Semantic Web application developers easily find related instances and query on various data sets. With the integrated ontology, we can also detect misuses of ontologies in the data sets and can recommend core properties for describing instances.

In future work, we plan to apply ontology reasoning methods to automatically detect and revise mistakes during the ontology merging process. Furthermore, we plan to automatically detect the undefined ranges of properties by analyzing the corresponding objects of properties in the data sets. We will test our framework with more data sets from the public linked data sets.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the Twentieth International Conference on Very Large Data Bases. pp. 487–499 (1994)
2. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. W3C Recommendation (2004), http://www.w3.org/TR/owl-ref/
3. Berners-Lee, T.: Linked Data - Design Issues (2006), http://www.w3.org/DesignIssues/LinkedData.html
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
5. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag, Heidelberg (2007)
6. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press (1998)
7. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool (2011)
8. Ichise, R.: An analysis of multiple similarity measures for ontology mapping problem. International Journal of Semantic Computing 4(1), 103–122 (2010)
9. Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z.: Ontology alignment for linked open data. In: Proceedings of the Ninth International Semantic Web Conference. LNCS, vol. 6496, pp. 402–417. Springer Berlin Heidelberg (2010)
10. Kohavi, R.: The power of decision tables. In: Proceedings of the Eighth European Conference on Machine Learning. pp. 174 – 189. Springer Verlag (1995)
11. Le, N.T., Ichise, R., Le, H.B.: Detecting hidden relations in geographic data. In: Proceedings of the 4th International Conference on Advances in Semantic Processing. pp. 61–68 (2010)
12. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
13. Meilicke, C., Völker, J., Stuckenschmidt, H.: Learning disjointness for debugging mappings between lightweight ontologies. In: Proceedings of the Sixteenth International Conference on Knowledge Engineering and Knowledge Management. LNCS, vol. 5268, pp. 93–108. Springer Berlin Heidelberg (2008)
14. Parundekar, R., Knoblock, C.A., Ambite, J.L.: Discovering concept coverings in ontologies of linked data sources. In: Proceedings of the Eleventh International Semantic Web Conference. LNCS, vol. 7649, pp. 427–443. Springer Berlin Heidelberg (2012)
15. Pedersen, T., Patwardhan, S., Michelizzi, J.: Wordnet::similarity: Measuring the relatedness of concepts. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence. pp. 1024–1025. Association for Computational Linguistics (2004)
16. W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C Recommendation (2012), http://www.w3.org/TR/owl2-overview/
17. Winkler, W.E.: Overview of record linkage and current research directions. Tech. rep., Statistical Research Division U.S. Bureau of the Census (2006)
18. Zhao, L., Ichise, R.: Graph-based ontology analysis in the linked open data. In: Proceedings of the Eighth International Conference on Semantic Systems. pp. 56–63. ACM (2012)
19. Zhao, L., Ichise, R.: Integrating ontologies using ontology learning approach. IEICE Transactions on Information and Systems Vol.E96-D(1), 40–50 (2013)