A Flexible System for Ontology Matching

Ngo DuyHoa, Zohra Bellahsene, Remi Coletta

LIRMM, Univ. Montpellier 2
34392 Montpellier, France
firstname.name@lirmm.fr

Abstract. Most of solutions provided by current ontology matching tools lack flexibility and extensibility namely for adding new matchers and dealing with users' requirements. In this paper, we present a system YAM++, which supports self-configuration, flexibility and extensibility in combining individual matchers. Moreover, it is more human-centered approach since it allows users to express their preference between precision and recall. A set of experiments over OAEI benchmark dataset demonstrate its effectiveness and efficiency in terms of quality of matching and flexibility of the system.
Keywords: Ontology matching, data mining, flexibility, self-configuration, cost-sensitive classification.

1 Introduction

Ontology matching is needed in many application domains. For example, the possibility of content-based query of the semantic Web depends only on the capacity of the system to find correspondences (mappings) between ontologies of the related information sources. Many diverse solutions of matching have been proposed so far; however, there is no integrated solution that is a clear success, which is robust enough to and flexible be the basis for future development, and which is usable by non expert users.

In this paper, we present our system YAM++, which supports self-configuration, flexibility and extensibility in combining individual matchers. To demonstrate the important of the flexibility in terms of system extensibility and user preference, let's us introduce two scenarios that frequently arise when people study schema and ontology matching.

In the first scenario, reseachers and developers of a matching system usually have to supplement new invented similarity metrics or update existing metrics with the new ones. According to [10], similarity metrics are also known as individual matchers. In both situations, developers must estimate the degree of contribution of these metrics and then find a suitable model to combine them. The estimation and combination models are normally tested carefully on existing "gold standard" datasets first, before applying them to real scenarios. In that case, a flexible system will help them to automatically deal with new metrics.

In the second scenario, imagine that users run a matching system to find all mapping pairs of entities between two ontologies. A matching system, generally,

outputs a list of candidate mappings and corresponding confident values. Users then must verify these mappings in order to remove incorrect ones. This process will not take much time because number of the suspect mappings is limited. Next, users need to find missing mappings which matching system did not discover. This process is very time consuming because it will be done manually on a huge number of candidate mappings. The manual effort of this phase is called post-match effort. Users may spend many hours or even few days to finish this work. Therefore, users desire to have a way to improve number of correct mappings in order to reduce post-match effort.

Based on these scenarios, the motivation of our system can be described as follows: Giving two ontologies represented in some ontology languages (N3, RDF, OWL, etc.), find a flexible approach to combine individual matchers with the following features: (i) achieving high matching quality result (*precision, recall and f-measure*), (ii) system's self-configuration, (iii) system's extensibility, (iv) generating a dedicated matcher according to the user's preference between precision and recall.

The remainder of this paper is organized as follows: In section 2, we describe our ontology matching system in detail. In Section 3, we present the results of experiments performed to highlight the main interesting features of our ontology matching tool. Section 4 contains the related work. Finally, Section 5 contains concluding remark about our system.

2 YAM++ Ontology Matching System

Our approach has been implemented in YAM++ - (not) Yet Another Matcher system for ontology matching. It follows the same approach used in YAM schema matching system [2]. However, the YAM++ aims to work with ontology matching, which is semantically richer than XML schema. For this purpose, we added new features such as:

- New similarity metrics working with different features (e.g. name, label, comments, relations) of ontologies' entities.
- New dictionary metrics based on different algorithms.
- New metrics based on information retrieval technique calculate similarity score between context and descriptive information of entities.
- Graphical user interface for setting parameters, displaying and verifying discovered mappings returned from system.

The main components of YAM++ system are depicted in Figure 1. It only requires as input, the set of ontologies to be matched. However, the user can also provide additional inputs, i.e., some preferences between precision and recall.

The **Knowledge Base** is a system repository, containing library of similarity metrics and library of learning models. It also stores list of gold standard datasets, which is a pair of ontologies with expert mappings between some of their entities built by domain experts.

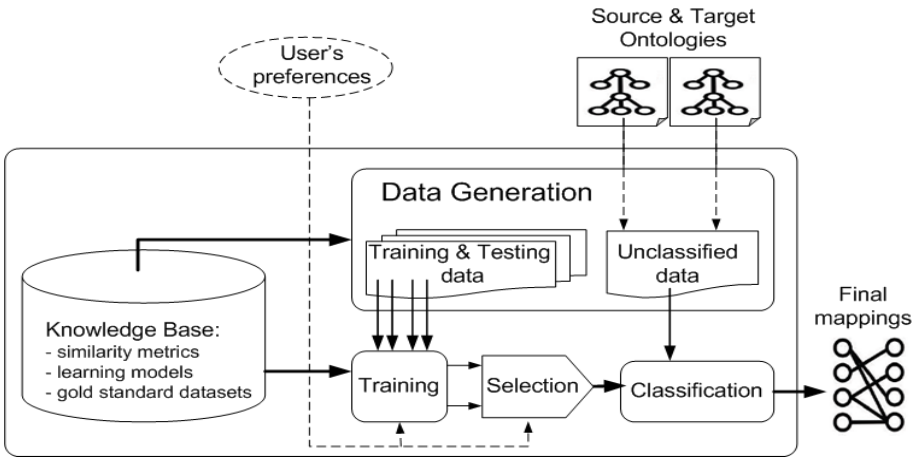

Fig. 1. YAM++ system architecture

The **Data Generation** module transforms gold standard datasets and input ontologies to learning (including training and testing) and unclassified data respectively. The idea is that each pair of entities (e_i, e_j) becomes a machine learning instance, which its features are the similarity score calculated by similarity metrics on (e_i, e_j). In training and testing data, the class of instances is determined by the confidence value of the corresponding pair (e_i, e_j) in the expert mappings set. For unclassified data, the class of instances is set to unknown value.

The **Training** module finds the optimal configuration for each learning model according to training data passed from **Data Generation** module. Besides, it can also take a user preference for either Precision or Recall in training process to generate classification models that favor this preference. The configuration process is automatic and transparent to users. The average performances (Precision, Recall, F-Measure) of all learning models achieved from running 10-fold cross validation and different testing data are temporarily saved for comparison purpose.

The **Selection** module by default will select a classification model, whose the obtained average F-Measure is highest. If user provides a preference between Precision and Recall, a classification model, which obtains the best result corresponding to this preference, is selected for next stage. In this paper, we call it **dedicated matcher** or **dedicated model**.

In the **Classification** module, a dedicated matcher predicts each instance in unclassified data by a predicted value. If the classification model is nominal, the predicted value is TRUE or FALSE, which means two entities corresponding with classified instance are matched or not.

Finally, these mappings are displayed in graphical user interface. Users can judge a mapping whether it is correct or not by their knowledge of ontologies'

domain. Users also can modify, remove incorrect mappings or add new mappings with the help of command operations appeared in system's menu (see Figure 2).

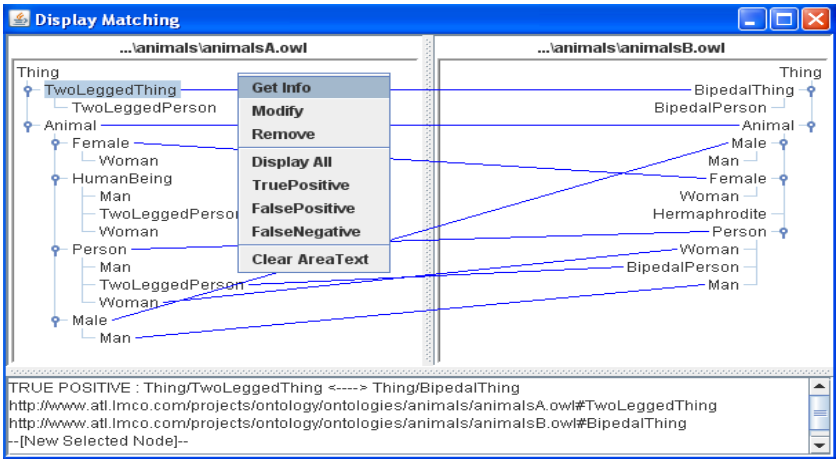

Fig. 2. User interface for mappings visualization

3 Experiments

In this section, we present the capabilities of our system using two experiments:

1. We show the flexibility and extensibility of our system in term of integrating new similarity metrics automatically and transparently to users.
2. We show another ability of generating a dedicated matcher based on the user's preference (promoting recall).

3.1 Experiment 1: Flexibility and Extensibility

For demonstration purpose, we run two scenarios and compare their performance quality.

– In the first scenario, we use a set of string metrics as individual matchers to calculate similarity value between entities based on entities' names and labels. These metrics are taken from open source code library SecondString[1], SimMetric[2]. These metrics are Levenstein, SmithWaterman, JaroWikler, Stoilos, QgramDistance, MongeEklan and Level2 metrics.

[1] http://secondstring.sourceforge.net/
[2] http://sourceforge.net/projects/simmetrics/

- In the second scenario, we add metrics working with dictionary WordNet[3] to exploit semantic features and metrics working with entities' description. We have implemented Lin and WuPalmer[7] algorithm for dictionary metrics. For comparing descriptive information, we construct a text corpus for each entity. Entity's corpus consists of its meta-data (name, labels, comments), meta-data of its related entities (sub-concepts, sub-properties, restricted properties, range). A Vector Space Model is constructed from these copora [8]. By using TF*IDF algorithm for term weighting, each corpus is transformed to a feature vector. The similarity score of two entities is calculated by cosine similarity of their feature vectors.
- In both scenarios we train different learning models such as: tree-based (J48, CART, ADTree, NBTree), probability-based (NaiveBayes, BayesNet), function-based (SMO, LibSVM, Logistic, MultiLayerPerceptron), instance-based (IBk, NNge, VFI). These models are taken from open source Weka[4] library. The gold standard datasets are taken from OAEI[5] and I3CON[6] repositories. In both scenarios, we do not set preference between Precision and Recall, so the criterion for selection is maximum F-Measure. The winner model after running selection process in both cases is DecisionTree J48 model. It means that the dedicated models used in Classification module are a trained J48 in both the scenarios.
- For comparison purpose, we run matching on set of datasets of OAEI 2009: {**#104, #203, #204, #205, #206, #201, #201-2, #201-4, #201-6, #201-8**}.

The observations from Figure 3 are:

- On datasets **#104, #203 and #204**, both scenarios have the same F-Measure ($\approx$ 1.0). This is because the linguistic information of these test ontologies is highly similar to with that of the reference ontology. In fact, entities' names of these ontologies are identical or are modified by some naming conventions. Therefore, adding the new metrics do not have a significant impact on the result.
- On datasets **#205 and #206** entities' names in test ontologies are replaced by synonym words or are translated in another language. Thank to using metrics based on dictionary and descriptive information, the achieved average F-Measure is increased **57%** from the first (**0.36**) to the second scenario (**0.93**).
- On datasets **#201, #201-2, #201-4, #201-6 and #201-8**, entities' names in test ontologies are replaced by random sequence symbols with **100%, 20%, 40%, 60% and 80%** respectively. However, entities are also described by labels and comments, so the achieved average F-Measure is increased **45%** from the first (**0.52**) to the second scenario (**0.97**).

[3] http://wordnet.princeton.edu
[4] http://www.cs.waikato.ac.nz/ml/weka/
[5] http://oaei.ontologymatching.org/2009/
[6] http://www.atl.lmco.com/projects/ontology/i3con.html

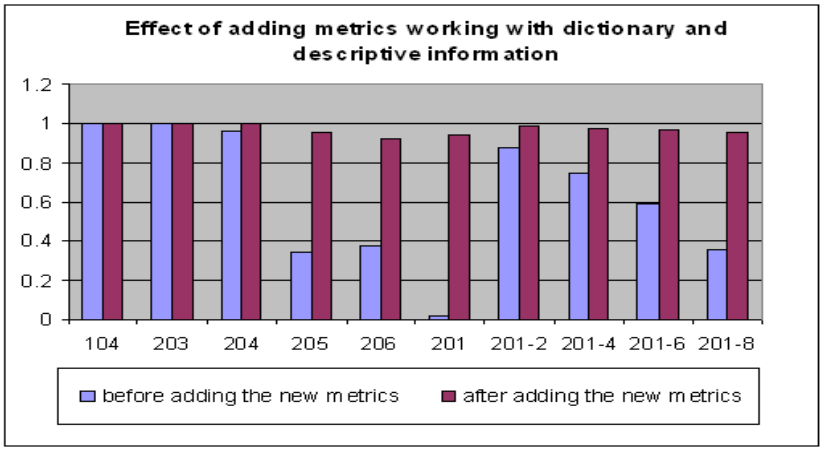

Fig. 3. Comparison on F-Measure of all datasets in the first and the second scenarios

The most interesting feature to note is that the process of reconfiguration system with new metrics is totally automatic and transparent to the users.

3.2 Experiment 2: Promoting Recall

Traditionally, the measure used to compute performance quality of matching tools, is the F-Measure: a combination of Precision (the ratio of correctly found correspondences (a.k.a true positive) over the total number of returned correspondences [5]) and Recall (the ratio of correctly found correspondences over the total number of expected correspondences [5]), in which precision and recall have the same weight. F-Measure makes sense when using matching tool as black box, without any user validation. But, most of the time, the user have to perform some post-match effort in order to discard some irrelevant and discover the missing mappings. In this experiment, we demonstrate the impact of user preference between Precision and Recall on post-match effort.

Technically, most classification models suffer from two errors during classifying: i) discovering an irrelevant correspondence (a.k.a. false positive) and ii) missing a relevant correspondence (a.k.a. false negative). The first error decreases precision while the second one decreases recall. In order to get better result in term of recall, we need to set the cost on false negative error higher than that on false positive error. This is a well-known issue called Cost-Sensitive Learning in Data Mining [4].

In order to deal with cost-sensitive learning, we use MetaCost and CostSensitiveClassification algorithms [11]. These algorithms belong to meta-learner class. They make a wrapper on base learning models in such a way that learning models effectively minimize cost . The preference between Precision and Recall is expressed by a proportion of the cost on false negative and the cost on false positive.

We perform our experiments with different proportion values, on the real datasets in OAEI 2009: {**#301, #302, #303, #304**}. The base learning models, the list of similarity metrics and training data are the same as described in the second scenario in the first experiment.

	proportion = 1	proportion = 5	proportion = 10	proportion ≥ 15
Total True Positive	122	127	**133**	**133**
Total False Positive	40	61	**83**	97
Total Undiscovered	19	14	**8**	**8**

Table 1. The effect of promoting Recall

For each proportion value, one dedicated matcher is generated. The observations from Table 1 are:

- By increasing the proportion value, the total of candidate mappings discovered as True Positive is increased. This advantage helps users to reduce time for discovering missing mappings.
- When the proportion is equal to 10, the total number of true positive mappings is maximum. After that, only the total number of false positive increases. This is a disadvantage, because users must to remove more irrelevant mappings.
- Notice that whatever the value we set for proportion, it always remains some matches we are not able to discover automatically.

As an example, when proportion is set to 10, the dedicated matcher discovers **11** (133 - 122) additional true positives, but **43** (83 - 40) additional false positives in comparison with the default matcher. In fact, the effort for manually removing an incorrect mapping is much less than the one for discovering a new correct mapping among **9949** pairs (total candidate mappings of 4 datasets). Therefore, by promoting recall, our system reduces user's post-match effort during the validation phase.

4 Related work

There are many studies on Ontology Matching [6],[5]. In this section, we only mention the closest ones that are based on machine learning approaches.

GLUE [1] is a well-known of learning-based ontology mapping system. GLUE uses a set of base learners to exploit different type of information from instances and taxonomy structures. Then, it uses a meta-learner to combine these base learners to achieve higher classification accuracy than any single base learner alone. The drawback of GLUE is that it requires a large number of instances associated with the nodes in taxonomies, whereas most ontologies do not contain these information. YAM++ is different with GLUE in that YAM++ uses machine learning approach to combine different individual matchers which exploit

different features of entities such as name, description and structure information. YAM++ does not exploit information of instance associated with entities.

Another systems using machine learning approach for ontology mapping such as APFEL [3] and [9]. Our approach and these systems are quite similar in the way of using machine learning approach to combine different similarity metrics. However, in YAM++, we use some other data mining techniques to help users reduce the post-match effort.

5 Conclusion

In this paper, we present a flexible system for ontology matching task that proves the following interesting features:

- Flexibility and extensibility in terms of combining individual matchers.
- Generating a dedicated matcher according to the user's preference.

We have developed a prototype which has been tested with the datasets of OAEI 2009 benchmark. Through these experiments, we have validated the features listed above.

References

1. AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies*, pages 385–404. 2004.
2. Fabien Duchateau, Remi Coletta, Zohra Bellahsene, and Renée J. Miller. Yam: a schema matcher factory. In *CIKM*, pages 2079–2080, 2009.
3. Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping ontology alignment methods with apfel. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, WWW '05, pages 1148–1149, New York, NY, USA, 2005. ACM.
4. Charles Elkan. The foundations of cost-sensitive learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.
5. Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
6. Yannis Kalfoglou and W. Marco Schorlemmer. Ontology mapping: The state of the art. In *Semantic Interoperability and Integration*, 2005.
7. Feiyu Lin and Kurt Sandkuhl. A survey of exploiting wordnet in ontology matching. In *IFIP AI*, pages 341–350, 2008.
8. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008.
9. Ming Mao, Yefei Peng, and Michael Spring. Ontology mapping: As a binary classification problem. *Semantics, Knowledge and Grid, International Conference on*, 0:20–25, 2008.
10. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
11. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.