Automatic Configuration Selection Using Ontology Matching Task Profiling*

Isabel F. Cruz[1], Alessio Fabiani[1], Federico Caimi[1], Cosmin Stroe[1],
and Matteo Palmonari[2]

[1] ADVIS Lab, Department of Computer Science, University of Illinois at Chicago, USA
{ifc,fabiani,fcaimi,cstroe1}@cs.uic.edu
[2] DISCo, University of Milan-Bicocca, Italy
palmonari@disco.unimib.it

Abstract. An ontology matching system can usually be run with different configurations that optimize the system's effectiveness, namely precision, recall, or F-measure, depending on the specific ontologies to be aligned. Changing the configuration has potentially high impact on the obtained results. We apply matching task profiling metrics to automatically optimize the system's configuration depending on the characteristics of the ontologies to be matched. Using machine learning techniques, we can automatically determine the optimal configuration in most cases. Even using a small training set, our system determines the best configuration in 94% of the cases. Our approach is evaluated using the AgreementMaker ontology matching system, which is extensible and configurable.

1 Introduction

Ontology matching is becoming increasingly important as more semantic data, i.e., data represented with Semantic Web languages such as RDF and OWL, are published and consumed over the Web especially in the *Linked Open Data (LOD)* cloud [14]. Automatic ontology matching techniques [10] are increasingly supported by more complex systems, which use a strategy of combining several *matching algorithms* or *matchers*, each taking into account one or more ontology features. Methods that combine a set of matchers range from linear combination functions [17] to matcher cascades [7, 10, 25], and to arbitrary combination strategies modeled as processes where specific matchers play the role of combiners [4, 16]. The choice of parameters for each matcher and of the ways in which the matchers can be combined may yield a large set of possible configurations. Given an *ontology matching task*, that is, a set of ontologies to be matched, an ontology engineer will painstakingly create and test many of those configurations manually to find the most effective one as measured in terms of precision, recall, or

* Research partially supported by NSF Awards IIS-0812258 and IIS-1143926 and by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-10-C-7061. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, AFRL, or the U.S. Government.

E. Simperl et al. (Eds.): ESWC 2012, LNCS 7295, pp. 179–194, 2012.

F-measure. Therefore, when given a new matching task, ontology engineers would like to start from their own set of available configurations and automatically determine the configuration that provides the best results without starting from scratch.

However, an ontology matching system that has been configured to match specific ontologies may not produce good results when matching other types of ontologies. For example, LOD ontologies vary widely; they can be very small (at the schema level), shallow, and poorly axiomatized such as GeoNames,[1] large with medium depth and medium axiomatization such as in DBpedia,[2] or large, deep, and richly axiomatized such as Yago.[3]

In this paper we propose a machine learning method that takes as input an ontology matching task (consisting of two ontologies) and a set of configurations and uses matching task profiling to automatically select the configuration that optimizes matching effectiveness. Our approach is implemented in the AgreementMaker [2] ontology matching system and evaluated against the datasets provided by the 2011 Ontology Alignment Evaluation Initiative (OAEI)[4]. We show that the automatically configured system outperforms the manually configured system. Furthermore, since the OAEI datasets are designed to test systems on a variety of ontologies characterized by heterogeneous features, they provide a rich and varied testbed. This testbed demonstrates the capability of our method to match many real-world ontologies and to achieve good effectiveness on new matching tasks without requiring manual tuning.

Although there are several machine learning approaches for ontology and schema matching, our approach differs from them in key aspects. Some approaches learn to classify mappings as correct or incorrect by combining several matchers [8] or similarity functions [13]. Others exploit user feedback to learn optimal settings for a number of parameters such as the similarity threshold used for mapping selection [25], or the weights of a linear combination of several matchers and the rate of iterative propagation of similarity [7]. Our approach learns how to select the optimal configuration for a matching task without imposing any restriction on the complexity of the adopted configurations, thus allowing for the reuse of fine-tuned configurations tested in previous ontology matching projects. We also achieve very good results with a limited training set by focusing on the feature selection problem, identifying fine-grained ontology profiling metrics, and combining these metrics to profile the ontology matching task.

The rest of this paper is organized as follows: Section 2 defines the problem of configuration selection and describes the configurations of the AgreementMaker system used in our experiments. Section 3 describes the proposed learning method to automatically select the best configuration and discusses the matching task profiling techniques adopted. Section 4 describes the experiments carried out to evaluate the approach. Section 5 discusses related work. Finally, Section 6 presents our conclusions.

2 Preliminaries

In this section we will explain some preliminary concepts, define the problem, and give a brief description of our AgreementMaker system and its configurations.

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

[2] http://www.dbpedia.org

[3] http://www.mpi-inf.mpg.de/yago-naga/yago/

[4] http://oaei.ontologymatching.org/2011/

2.1 Problem Definition

Ontology Matching (or *Ontology Alignment*) is defined as the process of finding correspondences between semantically related concepts in different ontologies [10]. Typically two ontologies are considered at a time—the *source* and the *target*—but the overall problem can involve several ontologies to be matched. The resulting correspondences are called *mappings*, a set of mappings is called an *alignment* and the correct alignment, as determined by domain experts, is called the *reference alignment* or *gold standard*. The algorithms used to discover the alignments are called *matchers*. A *basic matcher* takes into account a single aspect of the concepts to be matched, while a *complex matcher* is the result of combining and tuning basic matchers. The aggregation of matching results is called *combination*. The possible ways of combining results are mainly two: *series* and *parallel*. The former means that a matcher starts its computation based on the output result of another matcher, while the latter consists of taking the results from several matchers as input and providing a single output [4]. The resulting combination is called a *matcher stack*.

The word *configuration* refers to an already tuned matcher stack, usually organized in combination layers, whose number depends on the various series and parallel combinations that take place. Developing a configuration is a very complicated and time-consuming process that requires many experiments performed by a domain expert to define the matcher stack and the parameters for each matcher. That is, many choices must be made: which basic matchers to use, how to combine them, how many layers, how to set the thresholds and other parameters for each matcher. Therefore, it is clear that the exploration of the whole space of possible configurations is not a practical option.

It is also highly unlikely for the same configuration to perform well on totally different domains. We will approach this challenge by considering a small number of different configurations that have been manually developed for a variety of ontology matching tasks. This means that we do not address the problem of developing configurations, as we use the most successful ones that we have manually implemented with the AgreementMaker ontology matching system; instead, we focus on how to choose the best configuration for each task. That is, given two ontologies to be matched—an *ontology matching task*—and a set of configurations of a given ontology matching system, the problem consists of determining automatically the configuration that maximizes the quality of the output alignment, measured in terms of precision, recall, or F-measure (the weighted harmonic mean of precision and recall).

2.2 Matcher Configurations

The AgreementMaker system [2, 3, 4, 5] is an extensible ontology matching framework that has been expanded to include many types of matching algorithms so as to handle different matching scenarios. At the heart of the system is its ability to efficiently combine the results from several matching algorithms into one single and better alignment [4]. This capability allows us to focus on developing new matching algorithms, which can be combined with previously developed algorithms, with the objective of improving our results.

The configurations used in our system, which are shown as block diagrams in Figure 1, are derived from our experience of matching ontologies over the years. Our

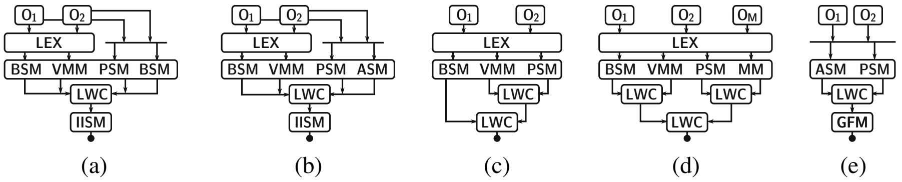


Fig. 1. The five configurations used in our system

work began with a general purpose configuration that consisted of syntactic matchers—the Base Similarity Matcher (BSM), the Parametric String Matcher (PSM), and the Vector-based Multi-word Matcher (VMM)—running in parallel and combined into a final alignment by the Linear Weighted Combination (LWC) Matcher [4]. We later extended our syntactic algorithms with lexicon lookup capabilities (LEX) and added a refining structural matcher—the Iterative Instance and Structural Matcher (IISM)—leading to the configuration shown in Figure 1(a). The configuration consists of running syntactic and lexical matching algorithms, combining their results, and then refining the combined alignment using a structural algorithm; this is a recurring pattern in all of our configurations.

Some ontologies may contain labels and local names that require more advanced string similarity techniques that are needed to perform syntactic matching. For this reason, our second configuration, which is shown in Figure 1(b), features the Advanced Similarity Matcher (ASM) [5]. This matcher extends BSM to find the similarity between syntactically complex concept labels.

As our work extended into matching biomedical ontologies [6] we found that the lexicon lookup capability of our algorithms became very important in those tasks. Such capability was especially useful because the biomedical ontologies include synonym and definition annotations for some of the concepts. For these types of ontologies, we use the configuration shown in Figure 1(c), which aggregates the synonyms and definitions into the lexicon data structure. There is more than one combination step so as to group similar matching algorithms before producing a final alignment.

An extension of the previous lexical-based configuration is shown in Figure 1(d). This configuration adds the Mediating Matcher (MM) to aggregate the synonyms and definitions of a third ontology, called the *mediating ontology*, into the lexicon data structure to improve recall. Finally, when matching several ontologies at a time, overall precision and runtime is more important. For this purpose we use a configuration, shown in Figure 1(e), which features the combination of two syntactic matchers and is refined by a structural matching algorithm that ensures precision and runtime efficiency.

3 Learning to Automatically Select the Best Configuration

Our approach can be sketched as follows: the matching task is profiled using several metrics, then the values for those metrics are used as input features to a machine learning algorithm that correlates the profiles to the performance of different configurations of the system. We are using a supervised learning approach and training the classifier on a subset of our dataset. Our goal is to use as small a subset as possible for training, as this reflects a real-world use of matching systems where users are able to provide only a very small subset of reference alignments.

Our proposed matching process follows the steps represented in Figure 2. First the pair of ontologies to be matched is evaluated by the matching task profiling algorithm. Based on the detected profile, a configuration is selected and the matcher stack is instantiated. Finally, the ontology matching step is performed and an output alignment is obtained.

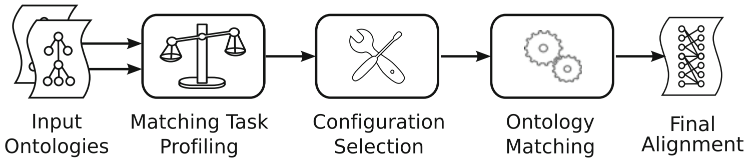


Fig. 2. Automatic configuration selection process

3.1 Ontology Profiling

Ontology profiling is the process of describing ontologies using a set of metrics. In particular, we include metrics that have been introduced in the context of ontology evaluation [26], where the goal is to evaluate the quality of an ontology. To the best of our knowledge, our approach is the first to apply these metrics in the context of ontology matching.

Since the aim of our work is to determine the most suitable system configuration for a given matching task, the first step is to characterize the task in terms of the two input ontologies. In particular, we use information about their classes and properties and combine the obtained profiles. This is a preprocessing phase with the goal of gathering information that is later used to discriminate between possible configurations. The metrics we use for ontology profiling can be evaluated by their expressiveness, clarity, and simplicity. A good metric should lend itself to be efficiently computed and to be correlated with some of the characteristics of the matchers that make up a configuration. The machine learning algorithm then exploits these correlations when building a classifier.

Our approach is unique because it is totally automatic, given a small training set. It makes no difference which matchers are part of a configuration, nor how they are combined. The machine learning algorithm will take care of summarizing the impact that some of the values of the metrics have on the performance of the matchers, without the need for explicit descriptions.

The metrics we used to profile the ontologies are shown in Table 1. The table also shows which ontology characteristics are taken into account by each metric (syntactic, lexical, structural, or instance). Some of these metrics are well known in the literature [8, 26], while others are being introduced in this paper. In what follows, C represents the set of classes in the ontology and T the set of terms, that is, the set of classes and properties. Therefore $|C|$ is the number of classes and $|T|$ is the number of terms.

Relationship Richness. The Relationship Richness (*RR*) of a schema is defined as the percentage of relations (object properties) at the schema level that are different from *subclassOf* relations. In the equation, *P* is the set of such relations and *SC* is the set of *subclassOf* relations [26].

Table 1. Ontology profiling metrics used by our system

Metric	Equation	Syntactic	Lexical	Structural	Instance
Relationship Richness	$RR = \frac{\|P\|}{\|SC\|+\|P\|}$			✔	
Attribute Richness	$AR = \frac{\|att\|}{\|C\|}$	✔		✔	
Inheritance Richness	$IR = \frac{\sum_{C_i \in C} \|H^C(C_l,C_i)\|}{\|C\|}$			✔	
Class Richness	$CR = \frac{\|C_i\|}{\|C\|}$				✔
Label Uniqueness	$LU = \frac{\|diff\|}{\|T\|}$	✔			
Average Population	$P = \frac{\|I\|}{\|C\|}$				✔
Average Depth	$D = \frac{\sum_{c_i \in C} D(C_i)}{\|C\|}$			✔	
WordNet Coverage	$WC_{i \in \{label,id\}} = \frac{\|covered_i\|}{\|C\|}$		✔		
Null Label and Comment	$N_{i \in \{label,comment\}} = \frac{\|NC_i\|}{\|T\|}$	✔			

Attribute Richness. Attribute Richness (*AR*) is defined as the average number of attributes (datatype properties) per class and is computed as the number of attributes for all classes (*att*) divided by the number of classes [26].

Inheritance Richness. Inheritance Richness (*IR*) describes the structure of an ontology. It is defined as the average number of subclasses per class [26]. The number of subclasses for a class C_i is defined as $|H^C(C_l, C_i)|$, where C_l is a subclass of C_i. An ontology with low *IR* has few children per class but many inheritance levels, while an ontology with high *IR* has few inheritance levels but many children per class.

Class Richness. Class Richness (*CR*) is defined as the ratio of the number of classes for which instances exist ($|C_i|$) divided by the total number of classes defined in the ontology [26].

Label Uniqueness. This metric captures the number of terms whose local name and label differ so as to determine whether we can find additional information in the term labels. We define Label Uniqueness (*LU*) as the percentage of terms that have a label that differs from their local name (*diff*).

Average Population. This metric provides an indication of the average distribution of instances across all the classes. Average Population (*P*) is defined as the number of instances $|I|$ divided by the number of classes [26].

Average Depth. This metric describes the average depth (D) of the classes in an ontology defined as the mean of the depth over all the classes C_i ($D(C_i)$).

WordNet Coverage. WordNet is a lexical database for the English language [20]. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synsets. WordNet is generally used in the ontology matching process to get information about the words contained in the attributes of a term. WordNet Coverage (*WC*) has been introduced as a feature that evaluates for each couple of terms whether none, one, or both of them can be found in WordNet [8]. Differently from previous approaches, in our system we compute *WC* as the percentage of terms with label or local name (*id*) present in WordNet (*covered*). We compute two WordNet Coverage metrics, one for local names and one for labels.

Null Label and Comment. Two other very important features of the ontology that we profile in our approach are the percentage of terms with no comment or no label, named respectively $N_{comment}$ or N_{label}. They are defined as the number of terms that have no comment ($|NC_{comment}|$) or no label ($|NC_{label}|$) divided by the number of terms.

3.2 Matching Task Profiling

Once all the metrics described in the previous section are computed for the two ontologies in the matching task, there are several methods we can use to compute the final matching task profile. We show that process in Figure 3. The first method consists of considering each metric value as a separate feature in the profile, which will be used as the feature vector for the classifier. In this case, the profile will contain just the values of the metrics for the source and target ontologies. We refer to this Source-Target profile configuration as *ST*. This approach, though, describes the ontologies singularly and does not give a compact summary of the pair. The second approach, instead, consists of combining the value pairs using a mathematical function. We experimented with two different combination functions: *Average* (*A*) and a newly defined function called *Feature Similarity* (*FS*). The former averages the values for a particular metric for both ontologies. It can be used to detect how much a particular characteristic is present in the matching task. For example, the average of the WordNet Coverage (*WC*) will be an indicator of how suitable a configuration relying on lexical matchers is for a particular matching task. When the average of *WC* is low, the model may learn to select a configuration without lexical matcher. Instead, the *FS* function evaluates how similar a characteristic is in the two ontologies. For example, when the average *WC* is high, *FS* explains whether *WC* is similar in the two ontologies or high in one and low in the other. Thus, the model may learn to select a configuration without lexical matcher even when *WC* is quite high on average, but very low in one ontology.

Given an ontology profiling metric m with two values, one for each ontology in the matching task, *FS* is defined as the ratio between the lower (m_L) and the higher (m_H)

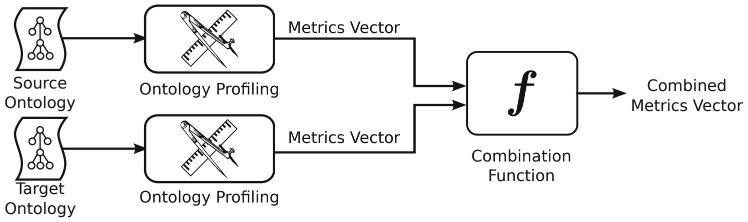


Fig. 3. Matching task profiling

of the two metric values, divided by the logarithm of their difference. When the two values are equal, the value of *FS* is one, while the more they differ, the closer to zero that value will be.

$$FS_m = \frac{m_L}{m_H[\log(m_H - m_L + 1) + 1]} \tag{1}$$

The concatenation of *A* and *FS* into a function *FS-A* provides a more expressive measure. It considers how much a particular aspect is present in each of the two ontologies as well as how much they share that aspect. Experiments described in Section 4 confirm this intuition showing that *FS-A* outperforms all the other functions.

3.3 Automatic Configuration Selection

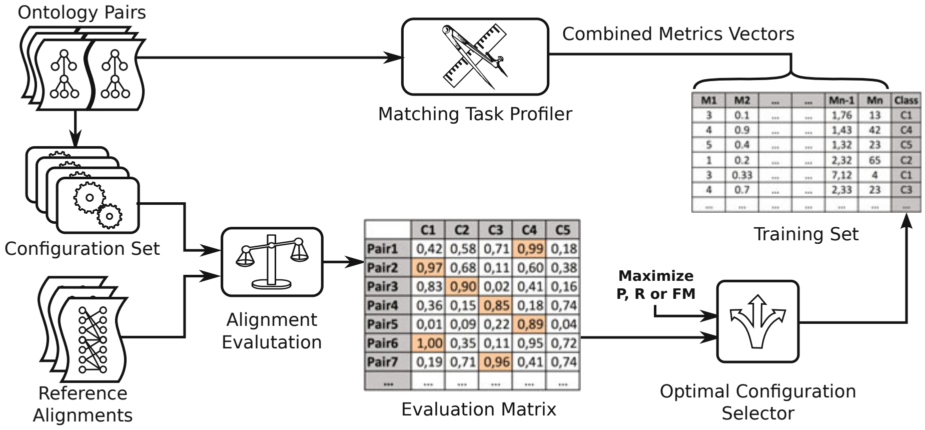
For each input ontology pair (matching task) we compute the metrics previously described and use them at runtime to predict the best configuration. The space of all the possible ontology pairs to be matched can be divided into a number of subspaces. We define those subspaces as the sets of ontology pairs sharing the same configuration as the best configuration for those matching tasks. From a machine learning point of view, each subspace corresponds to a class to be predicted.

Differently from rule-based approaches, our approach allows us to find correlations between metrics and the matching configurations without explicitly define them. No matter how complex a matcher is, its performance on the training set will allow the learning algorithm to determine its suitability to new matching tasks.

Supervised learning consists of techniques which, based on a set of manually labeled training examples, create a function modeling the data [19]. Each training example is composed of a feature vector and a desired output value. When the range of the output value is a continuous interval, it is called a regression problem. In the discrete case, it is called classification. The function created by the algorithm should be able to predict the output value for a new input feature vector. In our problem, the input feature vector is the result of matching task profiling while the output class is one of the possible configurations.

Building a robust classifier is not trivial: the main problem is how to generate a good training set. It should be a highly representative subset of the problem's data. In addition, the larger it is, the higher the quality of the classification will be. An important aspect is the distribution of the classes in the training set. It should reflect the nature of the data to be classified and contain an adequate number of examples for every class.

In order to generate our training set, the system goes through the steps shown in Figure 4. For each ontology pair in a matching task, we compute the metrics introduced in Section 3.1 and stored as data points in the training set. The system is then run with all the given configurations, each generating an alignment for every matching task. We then evaluate the precision, recall, and F-measure of each generated alignment and store the results in an *evaluation matrix*. For each matching task, the configuration that optimizes the precision, recall, or F-measure (depending on the users' needs) is chosen and stored as the correct class for the corresponding data points in the training set.



	C1	C2	C3	C4	C5
Pair1	0,42	0,58	0,71	0,99	0,18
Pair2	0,97	0,68	0,11	0,60	0,38
Pair3	0,83	0,90	0,02	0,41	0,16
Pair4	0,36	0,15	0,85	0,18	0,74
Pair5	0,01	0,09	0,22	0,89	0,04
Pair6	1,00	0,35	0,11	0,95	0,72
Pair7	0,19	0,71	0,96	0,41	0,74
...	...	...	...	...	...

M1	M2	...	...	Mn-1	Mn	Class
3	0.1	...	...	1,76	13	C1
4	0.9	...	...	1,43	42	C4
5	0.4	...	...	1,32	23	C5
1	0.2	...	...	2,32	65	C2
3	0.33	...	...	7,12	4	C1
4	0.7	...	...	2,33	23	C3
...	...	...	...	...	...	...

Fig. 4. Generation of the training set

We have tested this approach using many standard classifiers and including different feature vectors, which are compared in Section 4. The trained model is used by AgreementMaker to predict the correct configuration before starting a matching process. The system takes also into account the confidence value returned by the classification algorithm, which represents the certainty of the classifier's prediction. If this value is under a certain threshold the system ignores the classification and chooses the default configuration (Figure 1(a)).

4 Evaluation and Results

In this section we describe the experimental results obtained using the approach introduced in Section 3. Our approach has been evaluated using the AgreementMaker system with the configurations explained in Section 2.2. Our experiments have been run using 233 matching tasks with reference alignments provided in OAEI 2011.

4.1 Learning Evaluation

For our evaluation, we use several classifiers in order to gauge their impact on correctly classifying the matching tasks. The classifiers we use are well-known, each belonging to a different category: *k*-NN (Instance-based) [1], Naive Bayes (Probability-based) [12], Multilayer Perceptron (Neural Network) [11], and C4.5 (Decision Tree) [24].

For our first experiment, which is shown in Table 2, we perform a k-fold cross-validation with $k = \{2, 10\}$ using each classifier and comparing the obtained accuracy, defined as the percentage of correctly classified instances. 10-fold cross-validation is considered the standard for evaluating a learning algorithm while we also tested with 2-fold cross-validation to gauge the robustness of our approach, since we want as small a training set as possible. We experimented with all of our combination functions to generate the matching task profile. As can be seen from the table, a k-NN classifier (with $k = 3$) exhibits the best accuracy in all the tests. Furthermore, we can see that

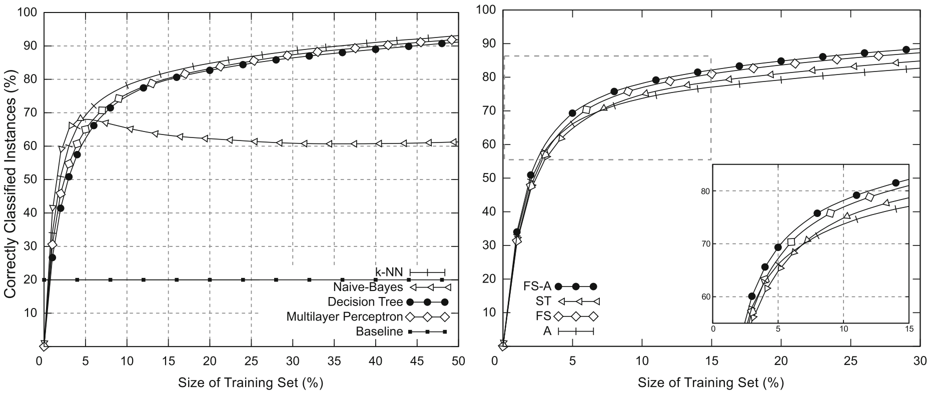
Table 2. Cross-validation accuracy of different classifiers and combination functions

Combination Function	Cross-validation	k-NN	Naive Bayes	Multilayer	C4.5
ST	10-fold	88.1%	55.0%	84.1%	85.9%
	2-fold	86.0%	57.0%	82.9%	83.6%
A	10-fold	85.7%	55.3%	84.5%	86.1%
	2-fold	82.9%	56.2%	82.6%	84.9%
FS	10-fold	87.6%	54.9%	84.9%	88.0%
	2-fold	85.8%	55.3%	82.6%	83.7%
FS-A	10-fold	**89.9%**	55.7%	88.1%	88.1%
	2-fold	**89.4%**	57.4%	83.8%	83.8%

the *FS-A* combination function has a significant impact on the overall results and in particular on making the approach more robust.

The graph of Figure 5(a) shows the accuracy that was obtained by varying the training set size and the classifier used. Each point is computed by averaging 100 runs. In each run, a random subset of matching tasks is selected for the training set and the model is evaluated against the remaining part of the dataset. Due to the obtained results, in this test we use *FS-A* as the combination function.

Naive Bayes is the worst performing classifier because the conditional independence assumption between the selected metrics does not hold. In other words, some of our metrics are interdependent. Since the dataset used is not big, instance-based classifiers perform slightly better than others. This is because they are able to learn quickly even from very small datasets. Other methods require a significant number of examples for each class to be capable of training robust models, while an instance-based classifier can make useful predictions using few examples per class. Therefore, this kind of classifier works well when limited data is available, as in our case. The other two classifiers,



(a) (b)

Fig. 5. Comparison of (a) classifiers and of (b) combination functions

Multilayer Perceptron and C4.5, show similar results, each being slightly worse than *k*-NN. The more the training set size increases, the more they close the gap with *k*-NN, confirming this characteristic of instance-based classifiers.

The results are also compared with a baseline. It is represented by the case in which the system randomly chooses between configurations. Since the number of configurations is five, the probability of selecting the right one is 20%, independently of the size of the training set. We believe these results are really valuable and encouraging, because even with a small training set our selection is able to outperform random selection substantially. As an example, when only 5% of the dataset is used for training, our method is able to choose the best configuration in seven cases out of ten. Moreover, in the other three cases, the results are only slightly worse than the best configuration. The two experiments above lead us to choose the *k*-NN classifier.

In Figure 5(b) we compare the combination functions, which we described in Section 3.2, as a function of the size of the training set. From the graphs in the figure it is clear that *FS-A* outperforms the other combination methods. Therefore we use *FS-A* in our following experiments.

4.2 OAEI Results

In the previous section, we have shown that our learning system is robust. In this section we use three tracks of the OAEI 2011 competition, namely Benchmark, Benchmark2, and Conference, to study the impact of our approach in terms of F-measure.

In order to guarantee an equally distributed and representative training set, we introduce the constraint that it should contain at least some representative instances for each of the five classes. Using this approach, our experiments were run using only 20% of the whole dataset as the training set, obtaining 94% accuracy. The remaining 6% of misclassified examples are further divided: 3.83% for which the algorithm chooses the second best configuration, and 2.17% for which it selects the third best one. The worst performing configurations (fourth and fifth) are never chosen. It is worth noting that even in the case where the algorithm fails to choose the best configuration, the chosen configuration is acceptable and in some cases is better than the one chosen manually.

Overall, our results were improved by using the automatic selection instead of the manual selection previously used in our system. This is because we are now able to choose the best configuration on a task-by-task basis, rather than on a track-by-track basis. Different configurations are used to match different tasks of the same domain, while selection performed by an expert is usually domain-specific.

In Table 3 we show the percentage of tasks in which our automatic approach outperforms the manual selection, for each of the considered tracks (Benchmark, Benchmark2, and Conference). We also show the average increase in F-measure for the tasks that were improved (the remaining ones were already optimal). Our new automatic selection leads to a significant improvement in all of the tracks, especially in the Conference track where more than half of the matching tasks showed improvement.

While in Table 3 we compare the results obtained over the whole testset, in Table 4 we show our improvements in terms of F-measure on some particularly interesting sub-tracks. We present an automatic versus manual selection comparison as before, but also versus an ideal classifier (i.e., a classifier for which the best configuration is always selected).

Table 3. Improvement of automatic vs. manual selection

	Benchmark	Benchmark2	Conference
Improved Tasks	8.0%	17.0%	57.0%
Avg. Gain/Task	3.4%	3.0%	12.7%

Table 4. Comparison between manual, automatic, and ideal selections

	Benchmark (301-304)	Benchmark2 (221-249)	Conference
Manual (M)	83.7%	82.4%	56.5%
Automatic (A)	86.7%	83.3%	61.0%
Ideal (I)	87.0%	83.6%	63.8%
$\Delta(A-M)$	3.0%	0.9%	4.5%
$\Delta(I-A)$	0.3%	0.3%	2.8%

We selected for this comparison a subset of the previously mentioned tracks, choosing the ones that we consider the most relevant, because they represent real-world examples. The sub-tracks are: Benchmark (301-304), Benchmark2 (221-249), and the Conference track as a whole. The Anatomy track is not included because it is composed of a single sub-track, which is correctly classified by both the automatic and manual selections. The table shows the average F-measures obtained by the selection approaches in these sub-tracks. We also show the difference between the automatic and manual selections ($\Delta(A-M)$) and between the ideal and automatic selections ($\Delta(I-A)$).

In Figure 6 we show the performance obtained by the manual, automatic, and ideal selections in the specified tasks of the Benchmark and Conference tracks. In most of these test cases the automatic selection chooses the correct configuration, that is, the automatically chosen configuration is also the ideal configuration. In some of these cases the manual selection chooses the correct configuration as well. An interesting case is provided by *confOf-ekaw*, where the three different modalities choose three different configurations. However, even in this case, the automatic selection chooses a better configuration than the one chosen manually.

5 Related Work

Several approaches have been proposed, whose objective is to improve the performance of ontology schema matching by automatically setting configuration parameters. An early approach considers a decision making technique that supports the detection of suitable matchings based on a questionnaire that is filled out by domain and matching experts [22]. In the continuation of this work, a rule-based system is used to rank a set of matchers by their expected performance on a particular matching task [21].

The RiMOM system profiles the ontology matching task by evaluating an overall lexical and structural similarity degree between the two ontologies [17]. These similarity measures are used to change the linear combination weights associated with a lexical and a structural matchers. These weights are adaptively set using matching task profiling, however only two metrics are used, a much smaller number than the number considered in this paper. Furthermore, the configuration cannot be changed.

The learning-based approaches for system configuration fall under two types: (1) learning to classify correspondences as correct/incorrect [8, 15, 23]; (2) learning optimal parameter values for a system [13, 16, 18]. As compared to these approaches, our contribution is that we learn to select the best configuration among a set of available configurations for each matching task. This introduces a new challenge: we must consider features describing the ontologies and their mutual compatibility as a whole and define ontology metrics for this purpose, both being novel contributions.

We now describe in more detail three of the machine-learning approaches [8, 13, 16]. The first approach learns the combination of several matchers, considered as black boxes [8]. The system uses a *mapping classifier* to label mappings as correct or incorrect based on a mapping profile, which encompasses features of the matchers, lexical and structural features of the concepts to be matched, and very simple ontology features. Our configuration selection is based exclusively on ontology features and embeds several more expressive features. It also has the following two advantages. First, we avoid executing every configuration of the system because, given a matching task, only the optimal configuration is selected, leading to significant savings of time and computational resources. Second, our classifier is trained on a small subset of the overall dataset. In particular, where in our experiments the overall dataset is split in a 1:4 ratio between training and evaluation data, the results for [8] are obtained with a ratio of 3:1 in the Benchmark and 4:1 in the Conference datasets. Our datasets are approximately the same size, but we use a much smaller training set, requiring less user effort. To the best of our knowledge, the minimization of the training set size has not been investigated by others. The second approach features a technique to automatically configure a schema matching system, which has been implemented in the eTuner application [16]. This approach is interesting because it is based on the use of synthetic data obtained by perturbing some features of the schemas to be aligned. However, it

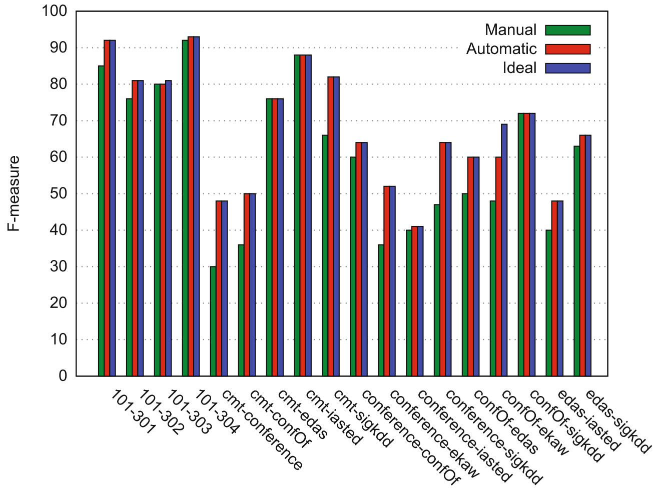


Fig. 6. Comparison between manual, automatic, and ideal selections for some of the track tasks

applies to matching relational schemas not to ontologies. The third approach combines several similarity metrics by learning an aggregated similarity function [13]. However, the degree to which the matching system is configured in our approach is significantly higher.

Finally, we mention some approaches that adopt active learning techniques in order to automatically tune the system on the basis of user feedback [7, 9, 25]. However, all these approaches learn to set system-specific parameters, such as the threshold [25], the optimal weights in a linear combination function, or the number of iterations of a similarity propagation matcher [7], while our system allows for configurations of arbitrary complexity.

6 Conclusions and Future Work

In this paper we made several contributions to the process of matching ontologies automatically. We proposed a learning approach to automatically select the best configuration from a predefined set. We used a set of metrics expressly chosen to describe ontologies and introduced combination functions to define a matching task profile. While other approaches can be more adaptive by not assuming an available configuration set and the effectiveness of our approach depends on the availability of a good configuration for that matching task, we show that using a relatively small number of heterogeneous configurations it is possible to achieve very good results. Furthermore, many ontology matching systems provide default configurations, which our approach can exploit.

Although we used the AgreementMaker ontology matching system, the strength of our approach is that it can produce excellent results for any system. The only requirements are: a set of matching tasks with reference alignment (for training), and a set of heterogeneous configurations covering diverse domains. Our work can be also extended by creating and evaluating new metrics and matching configurations. Another interesting research topic consists of adding a feedback mechanism to our approach [9].

References

[1] Aha, D.W., Kibler, D., Albert, M.K.: Instance-based Learning Algorithms. Machine Learning 6(1), 37–66 (1991)

[2] Cruz, I.F., Palandri Antonelli, F., Stroe, C.: Agreementmaker: Efficient matching for large real-world schemas and ontologies. PVLDB 2(2), 1586–1589 (2009)

[3] Cruz, I.F., Palandri Antonelli, F., Stroe, C.: Integrated Ontology Matching and Evaluation. In: International Semantic Web Conference, Posters & Demos (2009)

[4] Cruz, I.F., Palandri Antonelli, F., Stroe, C.: Efficient Selection of Mappings and Automatic Quality-driven Combination of Matching Methods. In: ISWC International Workshop on Ontology Matching (OM) CEUR Workshop Proceedings, vol. 551, pp. 49–60 (2009)

[5] Cruz, I.F., Stroe, C., Caci, M., Caimi, F., Palmonari, M., Antonelli, F.P., Keles, U.C.: Using AgreementMaker to Align Ontologies for OAEI 2010. In: ISWC International Workshop on Ontology Matching (OM). CEUR Workshop Proceedings, vol. 689, pp. 118–125 (2010)

[6] Cruz, I.F., Stroe, C., Pesquita, C., Couto, F., Cross, V.: Biomedical Ontology Matching Using the AgreementMaker System (Software Demonstration). In: International Conference on Biomedical Ontology (ICBO). CEUR Workshop Proceedings, vol. 833, pp. 290–291 (2011)

[7] Duan, S., Fokoue, A., Srinivas, K.: One Size Does Not Fit All: Customizing Ontology Alignment Using User Feedback. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 177–192. Springer, Heidelberg (2010)

[8] Eckert, K., Meilicke, C., Stuckenschmidt, H.: Improving Ontology Matching Using Meta-level Learning. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 158–172. Springer, Heidelberg (2009)

[9] Ehrig, M., Staab, S., Sure, Y.: Bootstrapping Ontology Alignment Methods with APFEL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 186–200. Springer, Heidelberg (2005)

[10] Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007) ISBN 3-540-49611-4

[11] Gardner, M.W., Dorling, S.R.: Artificial Neural Networks (the Multilayer Perceptron)–a Review of Applications in the Atmospheric Sciences. Atmospheric Environment 32(14-15), 2627–2636 (1998)

[12] John, G.H., Langley, P.: Estimating Continuous Distributions in Bayesian Classifiers. In: Conference on Uncertainty in Artificial Intelligence (UAI), pp. 338–345 (1995)

[13] Hariri, B.B., Sayyadi, H., Abolhassani, H., Esmaili, K.S.: Combining Ontology Alignment Metrics Using the Data Mining Techniques. In: ECAI International Workshop on Context and Ontologies, pp. 65–67 (2006)

[14] Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z.: Ontology Alignment for Linked Open Data. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 402–417. Springer, Heidelberg (2010)

[15] Köpcke, H., Rahm, E.: Training Selection for Tuning Entity Matching. In: International Workshop on Quality in Databases and Management of Uncertain Data (QDB/MUD), pp. 3–12 (2008)

[16] Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A.: eTuner: Tuning Schema Matching Software Using Synthetic Scenarios. VLDB Journal 16(1), 97–122 (2007)

[17] Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Transactions on Data and Knowledge Engineering 21(8), 1218–1232 (2009)

[18] Marie, A., Gal, A.: Boosting Schema Matchers. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 283–300. Springer, Heidelberg (2008)

[19] Marsland, S.: Machine Learning: an Algorithmic Perspective. Chapman & Hall/CRC (2009)

[20] Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: Introduction to WordNet: an On-line Lexical Database. International Journal of Lexicography 3(4), 235–244 (1990)

[21] Mochol, M., Jentzsch, A.: Towards a Rule-Based Matcher Selection. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 109–119. Springer, Heidelberg (2008)

[22] Mochol, M., Jentzsch, A., Euzenat, J.: Applying an Analytic Method for Matching Approach Selection. In: ISWC International Workshop on Ontology Matching (OM). CEUR Workshop Proceedings, vol. 225 (2006)

[23] Ngo, D., Bellahsene, Z., Coletta, R.: YAM++ Results for OAEI 2011. In: ISWC International Workshop on Ontology Matching (OM). CEUR Workshop Proceedings, vol. 814, pp. 228–235 (2011)

[24] Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1993)

[25] Shi, F., Li, J., Tang, J., Xie, G., Li, H.: Actively Learning Ontology Matching via User Interaction. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 585–600. Springer, Heidelberg (2009)

[26] Tartir, S., Budak Arpinar, I., Moore, M., Sheth, A.P., Aleman-Meza, B.: OntoQA: Metric-Based Ontology Quality Analysis. In: IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, vol. 9, pp. 45–53 (2005)