The Creation and Evaluation of iSPARQL Strategies for Matchmaking

Christoph Kiefer and Abraham Bernstein

Department of Informatics, University of Zurich, Switzerland
{kiefer,bernstein}@ifi.uzh.ch

Abstract. This research explores a new method for Semantic Web service matchmaking based on iSPARQL strategies, which enables to query the Semantic Web with techniques from traditional information retrieval. The strategies for matchmaking that we developed and evaluated can make use of a plethora of similarity measures and combination functions from SimPack—our library of similarity measures. We show how our combination of structured and imprecise querying can be used to perform hybrid Semantic Web service matchmaking. We analyze our approach thoroughly on a large OWL-S service test collection and show how our initial strategies can be improved by applying machine learning algorithms to result in very effective strategies for matchmaking.

1 Introduction

Imagine the following situation: Kate, a young and successful Semantic Web researcher, is trying to find a set of services to invoke for her latest project. Unfortunately, her department does not offer any useful semantically annotated services. She decides to look for some suitable services in a large database of crawled OWL ontologies that her department has gathered. Two issues arise: (1) as the task requires searching, she probably wants to use SPARQL; and (2), because there does not seem to be an ultimate, widely-accepted ontology for the domain, she will also have to consider approximate matches to her queries—a task for which statistics-based techniques from traditional information retrieval (IR) are well-suited.

In this paper, we address the task of matching a given user request with available information in a particular knowledge base (KB) assuming the lack of a perfect domain model—one of the basic underlying assumptions in Semantic Web research. More precisely, for a given input service request (*i.e.*, Kate's preference criteria), we aim to find all the relevant services that are closest to the query.

While many matchmaking algorithms have been proposed [11,14,17,18,24], we suggest not to build a specialized matchmaker (algorithm) but to build a matchmaker out of off-the-shelf components that embeds both structured (*i.e.*, logical) as well as statistical elements. Our method is based on iSPARQL queries (*i.e.*, *iSPARQL strategies*) [13], which enable the user to query the Semantic Web with a SPARQL extension that incorporates the notion of similarity—an approach often used in traditional IR. The strategies for matchmaking that

S. Bechhofer et al.(Eds.): ESWC 2008, LNCS 5021, pp. 463–477, 2008.

we developed and evaluate in this paper make use of a plethora of similarity measures and combination functions from SimPack—our library of similarity measures for the use in ontologies.[1]

We show how simple it is to find well-performing matchmaking strategies in an iterative and 'playful' procedure. Furthermore, we show how the performance of the initial matchmaking strategies can be greatly improved by applying standard machine learning algorithms such as regression, decision trees, or support vector machines (SVMs) to result in very effective strategies for matchmaking.

To evaluate our approach, we took a large collection of OWL-S services and compared the effectiveness of a number of simply constructed and learned/induced iSPARQL strategies in the search of the most effective strategies. We found that some simple-to-construct strategies performed surprisingly well, whilst a simple extension of those strategies based on machine learning outperformed even one of the most sophisticated matchmakers currently available.

The paper is structured as follows: next, we introduce the most important related work, before we give a brief overview of our iMatcher approach for matchmaking in Section 3. Section 4 discusses the performance measures and data sets used in the evaluation. The results of our experiments are presented in Section 5. We close the paper with a discussion of the results, our conclusions, and some insights into future work.

2 Related Work

In 1999, Sycara *et al.* [24] proposed LARKS—the Language for Advertisement and Request for Knowledge Sharing. The language defines a specification for the input and output parameters of services (among others) as well as for the constraints on these values. Their matchmaking process contains a *similarity filter* that computes the distances between service descriptions using TF-IDF [1] and other ontology-based similarity measures.

Three years later, Paolucci *et al.* [18] continued the work of [24] focusing on DAML-S.[2] One of the major differences between the LARKS and DAML-S approach is that the latter solely relies on logic and ontological reasoning.

Also related is the work of Di Noia *et al.* [17] who proposed a service matchmaking approach based on CLASSIC [4]. They discuss a purely logic-based implementation that matches service demands and supplies based on their explicit normal form (*i.e.*, demands and supplies are terminologically unfolded into their names, number restrictions, and universal role quantifications). The matchmaking algorithm then distinguishes between *potential* and *partial* matches of demands and supplies (*i.e.*, matches with no conflicts and matches with conflicting properties of demands and supplies).

In 2005, Jaeger *et al.* [11] presented the OWLSM approach for matching service inputs, service outputs, a service category, and user-defined service matchmaking criteria. The four individual matching scores are aggregated resulting in

[1] `http://www.ifi.uzh.ch/ddis/simpack.html`
[2] `http://www.daml.org/`

an overall matchmaking score. The result to the user is a ranked list of relevant services.

Recently, Klusch *et al.* [14] introduced the OWLS-MX matchmaker. As its name already implies, OWLS-MX focuses on the matching of services described in OWL-S, the successor of LARKS and DAML-S. The main focus of the matcher lies on a service profile's input and output (I/O) parameters. In comparison to the approaches proposed in [17,18], OWLS-MX uses both logic-based as well as IR-based matching criteria to identify services which match with a given query service. Specifically, OWLS-MX successively applies five different matchmaking filters: *exact*, *plug in*, *subsumes*, *subsumed-by*, and *nearest-neighbor*. Among those, the first three are purely logic-based, whereas *subsumed-by* and *nearest-neighbor* are hybrid as they incorporate a similarity measure to compute the syntactic similarity between query and service I/O concept terms. As such, the hybrid filters let some *syntactically similar*, but *logically disjoint* services be included in the answer set of a query, and thus, help improve query precision and recall. To compute the similarities, OWLS-MX particularly relies on similarity measures which have been shown to perform well in information retrieval [6].

A similar approach to [14] is described by Bianchini *et al.* [3], in which they presented the FC-MATCH hybrid matchmaking algorithm that is based on WordNet and the Dice coefficient [8].[3]

Finally, we would like to mention two studies that are similar to iSPARQL—the approach our matchmaker is based on. First, Corby *et al.* [7] introduced the conceptual graph-based Corese search engine that defines an RDF query language enabling both ontological and structural approximations.[4] Ontological approximation deals with distances between nodes in ontologies, whereas structural approximation is about structural divergences between the query and the queried RDF data set. Especially the latter allows the user to search for resources related by an arbitrary relations path between them. Second, Zhang *et al.* [27] presented the SPARQL-based Semplore system that combines structured querying with keyword searches using existing IR indexing structures and engines.

3 The iMatcher Approach

In this section we explain how our proposed matchmaking approach called *iMatcher* works. To that end, we first introduce the underlying iSPARQL technology and then explain the functionality of iMatcher itself.

3.1 Foundations: iSPARQL Matchmaking Strategies

The matchmaking approach we present in this paper is based on our previous work on iSPARQL [13]. We, therefore, succinctly review the most fundamental concepts. In iSPARQL we make use of ARQ property functions to apply similarity operators to SPARQL query elements.[5] The concept behind property

[3] WordNet lexical database of English, `http://wordnet.princeton.edu/`
[4] Conceptual graphs working draft, `http://www.jfsowa.com/cg/cgstand.htm`
[5] `http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions`

```
 1 PREFIX isparql: <java:isparql.>
 2 PREFIX profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>
 3
 4 SELECT ?p2 ?sim1 ?sim2 ?sim
 5 WHERE
 6   { # Basic graph pattern (BGP) matching part (lines 4-8)
 7     <P1> profile:name ?name1 ;
 8          profile:desc ?desc1 .
 9     ?p2  profile:name ?name2 ;
10          profile:desc ?desc2 .
11
12     # Virtual triple pattern matching part (lines 12-14)
13     IMPRECISE
14       { ?sim1 isparql:nameSimilarity ( ?name1 ?name2 ) .
15         ?sim2 isparql:textSimilarity ( ?desc1 ?desc2 ) .
16         ?sim  isparql:aggregate      ( 0.3 ?sim1 0.7 ?sim2 ) .
17       }
18   } ORDER BY DESC (?sim)
```

Listing 1.1. Example iSPARQL matchmaking strategy

functions is simple: whenever the predicate of a triple pattern is prefixed with a special name (*e.g.*, `isparql`), a call to an external similarity function is made and arguments are passed to the function (in our case by the object of the triple pattern). The similarity between the arguments is computed and bound to the subject variable of the triple pattern. As an example, consider the simple iSPARQL strategy in Listing 1.1 that matches the service profile $P1$ to the following RDF data set (note that $P1$, $P2$, and $P3$ stand for particular URLs, *e.g.*, `http://example.org/grocerystore_food_service.owls#GROCERYSTORE_FOOD_PROFILE`):

$$
\begin{aligned}
D = \{\ & (P1, \text{ profile:name}, \text{ “FoodService”}),\\
& (P1, \text{ profile:desc}, \text{ “Returns food of grocery store”}),\\
& (P2, \text{ profile:name}, \text{ “StoreService”}),\\
& (P2, \text{ profile:desc}, \text{ “This service returns store food”},\\
& (P3, \text{ profile:name}, \text{ “GroceryFoodService”}),\\
& (P3, \text{ profile:desc}, \text{ “A grocery service selling food”})\ \}
\end{aligned}
$$

We briefly explain the evaluation procedure of this strategy. Refer to [13] for a more elaborate description of this procedure. In a nutshell, the evaluation includes the following two steps: first, the basic graph pattern (BGP) matching part (lines 7–10) returns the Cartesian product of the sets of bindings from the defined variables; and second, the evaluation of the virtual triple patterns (lines 14–16) computes the similarities between the passed arguments and merges them with the solutions from BGP matching. Applying simple name and text similarity measures from SimPack—a library of similarity measures which iSPARQL uses—this could end up in the similarities $sc_{p_1,p_2} = 0.3{\cdot}0.571{+}0.7{\cdot}0.667 = 0.8095$ for $P1$ and $P2$, and $sc_{p_1,p_3} = 0.3 \cdot 0.8 + 0.7 \cdot 0.189 = 0.372$ for $P1$ and $P3$, being $P2$ the most accurate service profile for $P1$ (note that $sc_{p_1,p_1} = 1.0$).

It is important to note that the discovery phase of services is potentially very costly with our implementation as we do not optimize our iSPARQL strategies. Our queries likely result in the evaluation of expensive cross joins of triples as well as the (possibly) repeated computation of similarities in the evaluation of the virtual triple patterns. The first issue can be addressed with iSPARQL query

optimization, which we investigated in [2,22]. The second issue, the optimization of virtual graph patterns inside an `IMPRECISE` clause, can be addressed with similarity indexes to cache repeated similarity computations—an issue which we have not addressed so far.

3.2 The iMatcher Procedure

Our proposed matchmaking system is called *iMatcher*.[6] The "i" stands for *imprecise* emphasizing its ability for approximate matching using techniques from information retrieval (IR). iMatcher performs hybrid Semantic Web service matchmaking. That is, it uses both logic- and IR-based techniques to search for suitable services, which makes it similar to other hybrid systems such as OWLS-MX [14] and FC-MATCH [3]. On the one hand, iMatcher uses logical inferencing in ontologies to reason about the services under consideration, and on the other hand, it computes statistics about the data as well as makes use of indexes and weighting schemes to determine the degree of match between queries and services.

Specifically, iMatcher computes similarities between the query and the services in the KB. The more similar a service to a query, the more likely it is to be a correct result. As parameters, iMatcher is given a set of similarity measures SM, a data set of queries and correct answers T, and a learning algorithm R. To *train iMatcher*, it first computes a similarity-based training set T^{sim} that contains the similarities between all the queries and services in T for all measures in SM and an indication if the combination of the service and the query is correct (*i.e.*, a true positive) or not. It then applies the algorithm R to T^{sim} to learn/induce an induction model M. To *use iMatcher*, it computes the similarities (by the measures SM) of a given query q to all services in a given KB, then uses M to predict the combined similarity (or likelihood) of a match, and returns the answers in decreasing order of similarity.

As iMatcher is based on iSPARQL, any similarity measure defined within SimPack can be use for SM. For the learning algorithms R, any induction algorithm implementing the Weka interface can be employed.[7] As a training set T, iMatcher expects a training data set consisting of a knowledge base with services, a list of queries, and a file specifying which services are the correct answers for any given query.

For the evaluation we ran the results through a simple statistics handler that knows of the set of relevant services (true positives) for a given input query service. This information is used to compute the precision vs. recall figures that we show throughout our evaluation (see Section 5). Furthermore, iMatcher implements the *SME2Plugin* interface to initialize the matchmaker with a particular service knowledge base and to run queries.[8]

[6] `http://www.ifi.uzh.ch/ddis/imatcher.html`

[7] `http://www.cs.waikato.ac.nz/~ml/weka/`

[8] Used at the 1st International Semantic Service Selection Contest (S3) at ISWC 2007 (`http://www-ags.dfki.uni-sb.de/~klusch/s3/index.html`).

4 Performance Evaluation

In this section we first briefly review the performance measures we use in our evaluation and then describe our detailed experimental procedure.[9] Note that both the performance measures and the procedure are slight extensions of the ones introduced and, hence, validated by Klusch [14].

4.1 Performance Measures for Matchmaking

Precision and Recall. Probably the most often used performance measures for matchmaking are precision and recall [1]. Given a query q, precision Pr is the fraction of the answer set A_q (retrieved by the matchmaker) that is relevant to the query, whereas recall Re is the fraction of relevant documents R_q which have been retrieved, *i.e.*, $Pr_q = \frac{|Ra_q|}{|A_q|}, Re_q = \frac{|Ra_q|}{|R_q|}$, where Ra_q denotes the subset of relevant documents in A_q. As the evaluation of a single query is oftentimes not sufficient to make a statistically significant statement, many queries are involved and the averages of precision and recall computed.

Macro-Average. We are interested in *macro-averaging* precision and recall [20,21] over all queries, as it gives equal weight to each user query [16]. We introduce the set $L = \{0, 0.05, 0.1, \ldots, 1.0\}$ of 21 *standardized recall levels* [1] as our goal is to compute average precision vs. recall figures (see Section 5) for each investigated matchmaking strategy. Furthermore note that an interpolation procedure (aka *ceiling*) is necessary to compute precision and recall at each standardized recall level as each query likely has a different number of relevant services (*i.e.*, different R_q values). For each query q_i, $i \in \{1, \ldots, n\}$, macro-averaging computes precision and recall separately at each level j, where $j \in L$, and then computes the mean of the resulting n values, *i.e.*,

$$Pr_j^M = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|Ra_j^{q_i}|}{|A_j^{q_i}|} \qquad Re_j^M = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|Ra_j^{q_i}|}{|R_{q_i}|}$$

R-Precision. R-Precision RPr [1] is used in this paper as single value summary of the performance of a particular matchmaking algorithm. It is the fraction of relevant services which have been retrieved when considering only the first $|R_q|$ services of the answer set, denoted Ra_q^*, *i.e.*, $RPr_q = \frac{|Ra_q^*|}{|R_q|}$. We use R-Precision to compare two matchmaking algorithms A and B on a per-query basis, *i.e.*,

$$RPr_q^{A/B} = RPr_q^A - RPr_q^B$$

A positive result for $RPr_q^{A/B}$ highlights the fact that algorithm A is more effective than B for query q, and vice versa. A zero result denotes equal performance of both algorithms.

[9] For an elaborate treatment of these measures, refer to the textbook of Baeza-Yates and Ribeiro-Neto [1] as well as to the [16,21].

4.2 iMatcher Evaluation Procedure and Benchmarking Data Set

The evaluation procedure is the following: based on our experiences, we (1) create a set of iSPARQL strategies (see Section 3) that we assume will perform well; for each query, we (2) run our matchmaker to obtain a list of services ranked by their similarity to the query; given these ranked lists of services, we (3) compute macro-averages by considering the queries' relevance sets (their true answers) as explained in Section 4.1; finally, we (4) plot the results to visually examine the effectiveness of the created matchmaking strategies. By iteratively identifying and replacing weak parts of strategies with parts that improve the matchmaking performance, we are able to find well-performing strategies for the given task and data set.

For all our experiments we use *OWLS-TC v2*—an OWL-S Semantic Web service retrieval test collection as benchmarking data set.[10] The collection consists of 576 services from seven different domains. It specifies 28 queries with their relevance sets (true answers) allowing us to compute the aforementioned statistics for our matchmaker.

5 Experimental Results

The ultimate goal of our experiments was to demonstrate the ease of use of iMatcher to perform Semantic Web service matchmaking based on iSPARQL strategies. To that end, we evaluated three sets of strategies: (1) *primitive strategies*: evaluation of a set of simple, off-the-shelf strategies; (2) *induced strategies*: assessment of the quality of machine learning techniques for matchmaking; and (3) *customized strategies*: estimation of the improvements of iteratively improved, self-engineered strategies. We close our experiments with a short comparison of strategies from other systems.

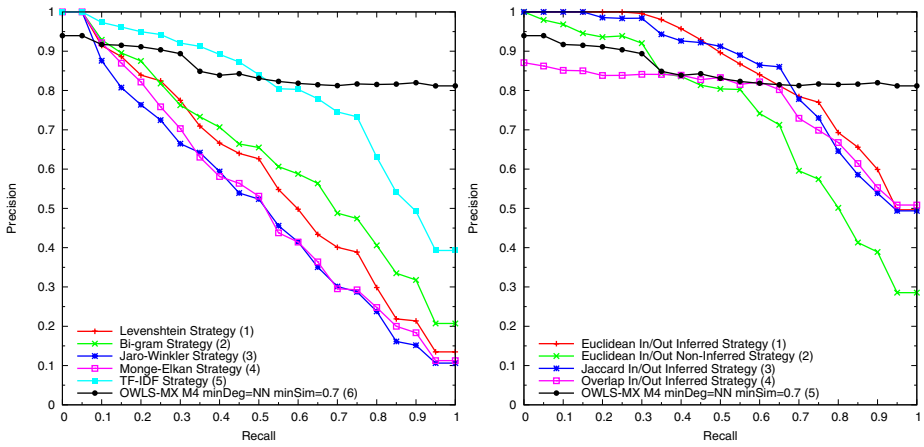
5.1 Primitive Strategies

String-based Strategies. We start our evaluation with the comparison of simple string-based similarity strategies for matchmaking (see Figure 1(a)). In addition, the results for *OWLS-MX M4* (measure no. 6) are shown on this and every subsequent precision vs. recall figure presented in this work. OWLS-MX M4 is reported to be the best-performing matchmaker variant of the OWLS-MX matchmaker [14]. It uses the extended Jaccard similarity coefficient to compare two services based on their sets of unfolded input/output concepts.[11] Furthermore, Figure 1(a) illustrates the results of the TF-IDF measure (no. 5) that compares services based on their service descriptions [1].

As the results in Figure 1(a) show, TF-IDF clearly outperforms all other measures in terms of precision and recall. It also outperforms OWLS-MX M4

[10] `http://projects.semwebcentral.org/projects/owls-tc/`

[11] In all our experiments, we used nearest-neighbor as minimum degree of match and a value of 0.7 as syntactic similarity threshold for OWLS-MX M4. These values were suggested by the authors of OWLS-MX to obtain good results for OWLS-TC v2.



(a) String-based strategies. (b) Vector-based strategies.

Fig. 1. Performance comparison of simple string- and vector-based strategies

until about half of the relevant services have been retrieved. Both strategies find all relevant services in the end (recall of 1.0), but OWLS-MX M4 with much higher, almost constant precision.

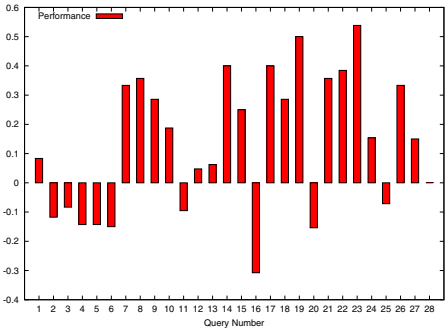
A clear performance trend is recognizable among the set of measures that syntactically compare the names of the services (measures no. 1–4). Among them, the Bi-gram string similarity measure (no. 2) performs slightly better than other prominent measures from this domain, such as the Levenshtein string similarity (no. 1) [15] and the Jaro measure (even with Winkler's reweighting scheme) [25].

Figure 2(a) additionally underscores the superiority of TF-IDF over the other strategies showing exemplarily a comparison with the Bi-gram measure on a per-query basis. For 19 out of 28 queries, comparing their textual descriptions with descriptions of services of the service KB turned out to be more efficient for matchmaking than comparing their service names. We speculate that these performance figures could be improved using even richer textual service descriptions.

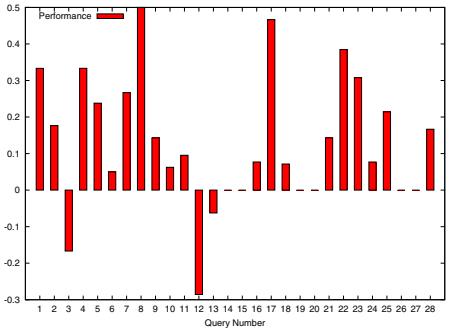
We learned from this evaluation that the simple TF-IDF full-text similarity measure is very well suited for matchmaking in OWLS-TC v2 to achieve good results. We, therefore, will reuse it in subsequent strategies. Furthermore, to compare service names, we will use the Bi-gram measure.

Vector-based Strategies. Next, we compare a set of simple vector-based strategies for matchmaking. The results are depicted in Figure 1(b). Basically, these strategies take the sets of service I/O concepts, logically unfold them (using the Pellet reasoner), transform them to vectors, and measure the similarity between those vectors using one of the vector similarity measures from SimPack.

More formally, let n, m be the number of input concepts of two particular web services a and b. The set of all input concepts I_a, I_b can be represented as binary vectors $\mathbf{x}_a$, $\mathbf{x}_b$ of size $|I_a| + |I_b| - |I_a \cap I_b|$, where $|I_a| = n$, $|I_b| = m$.



(a) TF-IDF vs. Bi-gram.



(b) J48 Decision Tree vs. Euclidean I/O inferred strategy.

Fig. 2. R-Precision comparison of selected matchmaking strategies

Consider, for example, the two services *surfing-beach-service* and *sports-town-service* from OWL-S TC v2 with their sets of unfolded input concepts $I_a = \{Surfing, Sports, Activity, Thing\}$ and $I_b = \{Sports, Activity, Thing\}$. The vector representation of these two sets is $x_a = [1, 1, 1, 1]^T$ and $x_b = [0, 1, 1, 1]^T$, that have, for instance, an extended Jaccard similarity of $\frac{|I_a \cap I_b|}{|I_a|+|I_b|-|I_a \cap I_b|} = 0.75$. Likewise, the similarity of the vectors representing the services' outputs is computed, which results in a value of 0.4. Averaging the two similarity scores, we obtain an overall similarity of 0.575 for the compared services a and b.

Figure 1(b) illustrates that simple measures such as the metric Euclidean vector distance are sufficient to achieve good results for I/O-based service matchmaking.[12] The extended Jaccard measure is only slightly outperformed by the simple Euclidean distance. The figure also shows that the Overlap measure fulfills the matchmaking task less precisely than the Euclidean and the extended Jaccard measure for about 65% of retrieved services. After that, its performance is comparable to the other measures.

It is interesting to observe the remarkable influence of ontological reasoning over the I/O concepts on the matchmaking task. Comparing the Euclidean measure with reasoning support turned on (measure no. 1 in Figure 1(b)) vs. the same measure without reasoning (no. 2) evidently shows that enabling reasoning boosts performance about 20% for the given matchmaking task. As a consequence, we will use reasoning in all subsequent experiments.

5.2 Machine-Learned Strategies

As we showed in our previous work [12], techniques from machine learning are well-suited for Semantic Web service discovery. Hence, we intend to also test their applicability for Semantic Web service matchmaking.

[12] The Euclidean distance d is converted into a similarity score by $\frac{1}{1+d}$.

Table 1. T^{sim} result table to learn a matchmaking strategy

Query	Service	Bi-gram	TF-IDF	EuclidIn	EuclidOut	TP(q,sv)
q_1	sv_1	$sc^{xs}_{q_1,sv_1}$	$sc^{tfidf}_{q_1,sv_1}$	$sc^{ei}_{q_1,sv_1}$	$sc^{eo}_{q_1,sv_1}$	1
...	...	...	...	...	...	...
q_1	sv_m	$sc^{xs}_{q_1,sv_m}$	$sc^{tfidf}_{q_1,sv_m}$	$sc^{ei}_{q_1,sv_m}$	$sc^{eo}_{q_1,sv_m}$	0
...	...	...	...	...	...	...
q_n	sv_1	$sc^{xs}_{q_n,sv_1}$	$sc^{tfidf}_{q_n,sv_1}$	$sc^{ei}_{q_n,sv_1}$	$sc^{eo}_{q_n,sv_1}$	1
...	...	...	...	...	...	...
q_n	sv_m	$sc^{xs}_{q_n,sv_m}$	$sc^{tfidf}_{q_n,sv_m}$	$sc^{ei}_{q_n,sv_m}$	$sc^{eo}_{q_n,sv_m}$	0

To that end, we induced four different machine learning models using algorithms from Weka[13] and LibSVM[14] as explained in 3.2. From Weka, we chose a linear regression, a logistic regression, and a J48 decision tree learner for the prediction if a service is a true answer to a query. From LibSVM, we chose the support vector regression model (ϵ-SVR) with an RBF kernel and cross-validation/grid-search done as recommended in [10] to search for the best parameter setting for the RBF kernel. For a more detailed discussion of these algorithms, refer to the textbook of Witten and Frank [26] for Weka and to Chang and Lin [5] for LibSVM.

For the four similarity measures *Bi-gram*, *TF-IDF*, *EuclidIn*, and *EuclidOut*, this resulted in T^{sim} shown in Table 1. From this table, we induced the models M, whilst ignoring the columns *Query* and *Service*, using the aforementioned algorithms from Weka and LibSVM.

The resulting iSPARQL strategy from Weka's linear regression model (its `IMPRECISE` clause) is shown in Listing 1.2. The model defines a weighted, linear combination of the input features (the similarity scores) for the prediction of the membership of a service to a query's relevance set. The weights learned for this particular model are given on the last two lines in Listing 1.2. The final similarity score sc is computed by aggregating/combining the individual scores, *i.e.*, $sc = \sum_{i \in SM} w_i \cdot sim_i$, where $SM = \{Bi\text{-}gram, TF\text{-}IDF, EuclidIn, EuclidOut\}$.

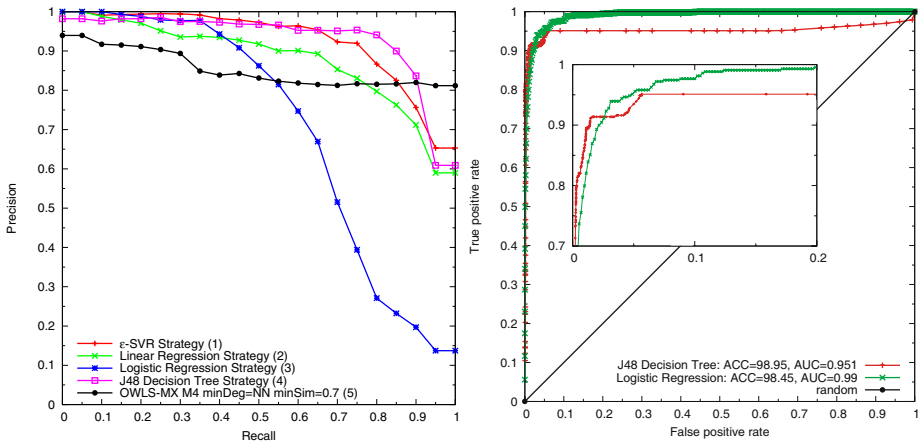
The results in Figure 3(a) show that the ϵ-SVR strategy outperforms all other strategies in terms of precision until about 65% of the relevant services have been retrieved on average. Then, the decision tree takes the lead until about 90% retrieved relevant services, before it gets again slightly outperformed by ϵ-SVR. Linear and logistic regression perform worse than the non-linear models.

The accuracy of the prediction for the J48 learner is 98.95% and 98.45% for the logistic regression learner. Note that the sole use of accuracies is, however, misleading, as they are heavily dependent on the prior of the data set. Therefore, Figure 3(b) graphs the Receiver Operating Characteristics (ROC) and the area under the ROC-curve (AUC; in legend) that both provide a prior-independent approach for comparing the quality of a predictor [19] for two of the chosen

[13] `http://www.cs.waikato.ac.nz/~ml/weka/`
[14] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

```
IMPRECISE
  { ?sim1 isparql:bigram     (?serviceName ?queryName) .
    ?sim2 isparql:tfidf      (?serviceDescription ?queryDescription) .
    ?sim3 isparql:euclidIn   (?serviceProfile ?queryProfile) .
    ?sim4 isparql:euclidOut (?serviceProfile ?queryProfile) .
    # learned combination of four different strategies
    ?sim  isparql:aggregate (0.0443 ?sim1 0.785 ?sim2 0.481 ?sim3
                             0.146 ?sim4 -0.158 1.0)
  }
```

Listing 1.2. Machine-learned strategy using a linear regression model



(a) Learned/Induced strategies. (b) ROC-curves.

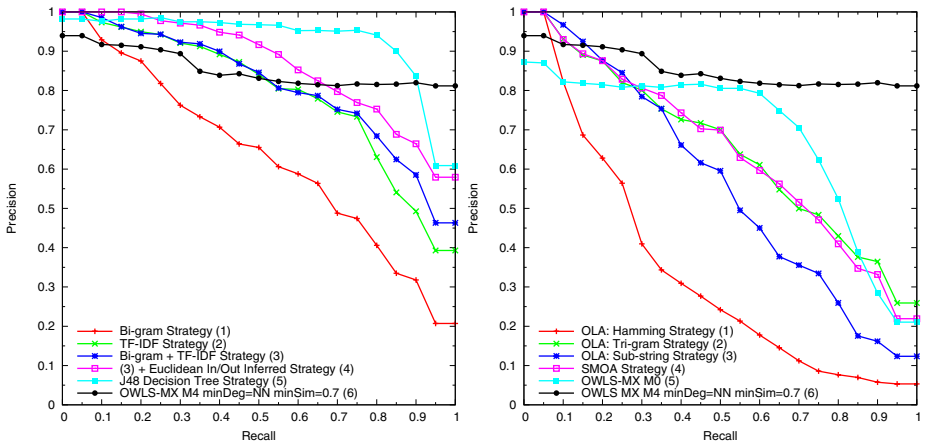
Fig. 3. Performance comparison of learned matchmaking strategies

methods. The x-axis shows the false positive rate and the y-axis the true positive rate. AUC is, typically, used as a summary number for the curve. A random class assignment (either YES or NO) is also shown as a line form the origin (0,0) to (1,1) and the ideal ROC-curve would be going from the origin straight up to (0,1) and then to (1,1).

Note that with this very simple approach we are able to clearly outperform OWLS-MX M4 in terms of precision and recall for almost 90% of relevant services. Also, the performance on a per-query basis illustrates the superiority of the machine learned strategies as plotted exemplarily in Figure 2(b) for the J48 decision tree strategy and the simpler vector-based Euclidean I/O strategy. We, therefore, conclude that the combination of logical deduction and statistical induction produces superior performance over logical inference only, which can be easily achieved with our approach.

5.3 Customized Strategies

As claimed in the introduction, iMatcher enables the creation of efficient matchmaking strategies in an iterative and 'playful' procedure. To point this out,



(a) Hand-crafted/Customized strategies. (b) Other system's strategies.

Fig. 4. Performance comparison of customized matchmaking strategies and strategies from other systems

Figure 4(a) summarizes the results for four iteratively improved strategies. Starting with the very simple name-comparing strategy Bi-gram (measure no. 1), we successively can create better strategies by first applying TF-IDF (no. 2), then combining 1 and 2 (= no. 3), followed by no. 4 that additionally takes service inputs and outputs into account, and last, by using machine learning to result in the best-performing strategy in this experiment.

5.4 Strategies from Other Systems

To close our experimental section, we succinctly compare the results of three other systems: (1) the ontology alignment tool OLA (OWL-Lite Alignment) [9]; (2) OWLS-MX [14]; and (3), the string metric for ontology alignment (SMOA) [23]. Note that two of these tools were initially created for ontology alignment rather than for matchmaking. However, as they also involve similarity measures to find correspondences in different ontologies, they can also be used by iMatcher.

The results in Figure 4(b) show that the Hamming strategy from OLA to compare the names of services is by far the most inaccurate approach for matchmaking. All of the other measures have much higher performance. The SMOA strategy behaves very similar to OLA's Tri-gram strategy. Finally, a comparison of OWL-S MX M0 that performs purely logic-based matchmaking with OWLS-MX M4 again illustrates the usefulness of taking into account simple methods from IR to improve precision and recall for Semantic Web service matchmaking.

6 Discussion and Limitations

Clearly, our empirical results are limited to an artificial data set, namely OWL-S TC v2. Therefore, the results must be taken with a pinch of salt. Given,

however, the very low number of comparable test collections publicly available, this limitation is rather natural and must be accepted. Similarly, we note that focusing only on OWL-S as service description language is a limitation of our experimental setup and not of our approach itself.

Particularly interesting is the high influence of the IR techniques on matchmaking performance. The strategies that exploit textual information of the services turned out to be very effective, which underlines again the importance of the statistics-based approaches for matchmaking. On the other hand, as can be seen from Figure 1(b), strategies that involve reasoning over the input data can boost matchmaking performance by a factor of about 20%. It is interesting to observe that the combination of both approaches (see measure no. 4 in Figure 4(a)) is superior to each of the individual approaches. Whilst this is not a new finding, it emphasizes, nevertheless, the importance of combining statistical inference with logical deduction, which can also be concluded from our machine learning experiments in Section 5.2.

Of course, creating customized indexes for keyword search and document weighting (*e.g.*, for TF-IDF) involves a pre-processing step of the data, which might be costly depending on the size and dynamics of the data set.

As mentioned earlier, we did not yet consider iSPARQL optimization techniques. Besides the work achieved for SPARQL basic graph pattern optimization through selectivity estimation [2,22], we did, however, experiment with various similarity index structures, whose elaborate treatment we postpone to future work. This is especially important if our approach should be scalable and applicable to data sets which are much larger than the one used in this paper.

Another issue is the approximate matching procedure of iMatcher as opposed to a formal approach. Obviously, both approaches have their important role. Formal matching procedures ensure a correct match, which is especially important when, for example, automatically finding a service that shall be invoked without any adaptation. The formal approaches, however, might be too restrictive in open domains, where an exact match cannot be assumed [14]. Approximate matching is more suitable when an exact match cannot be found, when the formal approach provides too many answers, or when some adaptation procedure would adapt the "calling" code to the found service before invoking it. Also, approximate matching raises the issue of a threshold under which an answer should not be considered. In our evaluation we assumed that the caller would like an answer in any case and would like to examine the most suitable result. In other scenarios such a threshold would, however, be appropriate.

7 Conclusions and Future Work

We presented a new approach to perform Semantic Web service matchmaking based on iSPARQL strategies. We showed how our combination of structured and imprecise querying can be used to perform hybrid Semantic Web service matchmaking.

We evaluated our approach by analyzing a multitude of different matchmaking strategies that make use of the logical foundations of the Semantic Web, apply techniques from traditional information retrieval, involve machine learning, or employ a combination of the three to find very effective strategies for matchmaking. Our empirical results suggest that simple strategies are oftentimes sufficient to obtain good results. However, using weighted and learned combinations of simple measures boosts matchmaking performance even further.

It is left to future work to analyze iMatcher's behavior for different Semantic Web data sets, such as OWLS-S TC v2.2, and service description formats, such as WSMO[15] or SAWSDL.[16]

Finally, coming back to the introductory example, the question is, of course, what Kate would say to these results? Given our experimental results, iMatcher seems to be a practical and easy to use tool to help Kate solving the requirements of her latest project, ensuring she will receive good grades for her work.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments, and Matthias Klusch for some interesting discussions regarding OWLS-MX and iMatcher.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, Reading (1999)
2. Bernstein, A., Kiefer, C., Stocker, M.: OptARQ: A SPARQL Optimization Approach Based on Triple Pattern Selectivity Estimation. Technical Report IFI-2007.02, Department of Informatics, University of Zurich (2007)
3. Bianchini, D., Antonellis, V.D., Melchiori, M., Salvi, D.: Semantic-Enriched Service Discovery. In: ICDEW, pp. 38–47 (2006)
4. Borgida, A., Brachman, R.J., McGuinness, D.L., Resnick, L.A.: CLASSIC: A Structural Data Model for Objects. In: SIGMOD, pp. 58–67 (1989)
5. Chang, C.-C., Lin, C.-J.: LIBSVM—A Library for Support Vector Machines (2001), Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
6. Cohen, W.W., Ravikumar, P., Fienberg, S.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: IJCAI Workshop, pp. 73–78 (2003)
7. Corby, O., Dieng-Kuntz, R., Gandon, F., Faron-Zucker, C.: Searching the Semantic Web: Approximate Query Processing Based on Ontologies. Intelligent Systems 21(1), 20–27 (2006)
8. Dice, L.R.: Measures of the Amount of Ecologic Association Between Species. Ecology 26(3), 297–302 (1945)
9. Euzenat, J., Loup, D., Touzani, M., Valtchev, P.: Ontology Alignment with OLA. In: EON, pp. 60–69 (2004)

[15] `http://www.wsmo.org/`
[16] `http://www.w3.org/2002/ws/sawsdl/`

10. Hsu, C.-W., Chang, C.-C., Lin, C.-J.: A Practical Guide to Support Vector Classification (2007)
11. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G., Liebetruth, C., Geihs, K.: Ranked Matching for Service Descriptions using OWL-S. In: KiVS, pp. 91–102 (2005)
12. Kiefer, C., Bernstein, A., Lee, H.J., Klein, M., Stocker, M.: Semantic Process Retrieval with iSPARQL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 609–623. Springer, Heidelberg (2007)
13. Kiefer, C., Bernstein, A., Stocker, M.: The Fundamentals of iSPARQL: A Virtual Triple Approach for Similarity-Based Semantic Web Tasks. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC 2007. LNCS, vol. 4825, Springer, Heidelberg (2007)
14. Klusch, M., Fries, B., Sycara, K.: Automated Semantic Web Service Discovery with OWLS-MX. In: AAMAS, pp. 915–922 (2006)
15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
16. Lewis, D.D.: Evaluating Text Categorization. In: HLT Workshop, pp. 312–318 (1991)
17. Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: A System for Principled Matchmaking in an Electronic Marketplace. IJEC 8(4), 9–37 (2004)
18. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
19. Provost, F., Fawcett, T.: Robust Classification for Imprecise Environments. Machine Learning 42(3), 203–231 (2001)
20. Raghavan, V.V., Bollmann, P., Jung, G.S.: Retrieval System Evaluation Using Recall and Precision: Problems and Answers. In: SIGIR, pp. 59–68 (1989)
21. Sebastiani, F.: Machine Learning in Automated Text Categorization. ACM Computing Surveys 34(1), 1–47 (2002)
22. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In: WWW (2008)
23. Stoilos, G., Stamou, G., Kollias, S.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
24. Sycara, K., Klusch, M., Widoff, S., Lu, J.: Dynamic Service Matchmaking Among Agents in Open Information Environments. SIGMOD Rec. 28(1), 47–53 (1999)
25. Winkler, W.E., Thibaudeau, Y.: An Application of the Fellegi-Sunter Model of Record Linkage to The 1990 U.S. Decennial Census. Technical report, U.S. Bureau of the Census (1987)
26. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005)
27. Zhang, L., Liu, Q., Zhang, J., Wang, H., Pan, Y., Yu, Y.: Semplore: An IR Approach to Scalable Hybrid Query of Semantic Web Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC 2007. LNCS, vol. 4825, pp. 653–665. Springer, Heidelberg (2007)