Boosting Schema Matchers

Anan Marie and Avigdor Gal

Technion – Israel Institute of Technology
32000, Israel
{sananm@cs,avigal@ie}.technion.ac.il

Abstract. Schema matching is recognized to be one of the basic operations required by the process of data and schema integration, and thus has a great impact on its outcome. We propose a new approach to combining matchers into ensembles, called Schema Matcher Boosting (SMB). This approach is based on a well-known machine learning technique, called boosting. We present a boosting algorithm for schema matching with a unique ensembler feature, namely the ability to choose the matchers that participate in an ensemble. SMB introduces a new promise for schema matcher designers. Instead of trying to design a perfect schema matcher that is accurate for all schema pairs, a designer can focus on finding better than random schema matchers. We provide a thorough comparative empirical results where we show that SMB outperforms, on average, any individual matcher. In our experiments we have compared SMB with more than 30 other matchers over a real world data of 230 schemata and several ensembling approaches, including the Meta-Learner of LSD. Our empirical analysis shows that SMB improves, on average, over the performance of individual matchers. Moreover, SMB is shown to be consistently dominant, far beyond any other individual matcher. Finally, we observe that SMB performs better than the Meta-Learner in terms of precision, recall and F-Measure.

1 Introduction

Schema matching is recognized to be one of the basic operations required by the process of data and schema integration [18]. Due to its cognitive complexity, schema matching has been traditionally considered to be AI-complete, performed by human experts [12]. Over the years, a significant body of work was devoted to the identification of *schema matchers*, *e.g.*, [4, 11, 19]. The main objective of schema matchers is to provide *schema matchings* that will be effective from the user point of view, yet computationally efficient. Over the years, the main research emphasis was on devising better and more effective matchers.

Choosing among schema matchers is far from being trivial. First, the number of schema matchers is continuously growing, and this diversity by itself complicates the choice of the most appropriate tool for a given application domain. Second, as one would expect, empirical analysis shows that there is no (and may never be) single dominant schema matcher that performs best, regardless of the data model and application domain [10]. Therefore, several tools, *e.g.*, [4,

7, 11, 15, 20] have combined different schema matchers to determine the similarity between concepts. The idea is appealing since an ensemble of complementary matchers can potentially compensate for the weaknesses of each other.

We propose a new approach to combining matchers into ensembles, called Schema Matcher Boosting (SMB), based on a well-known machine learning technique, called boosting [24]. SMB introduces a new promise for schema matcher designers. Instead of trying to design a perfect schema matcher, a designer can instead focus on finding better than random schema matchers. We show in Section 2.2 that this property is strongly tied with the monotonicity principle, as introduced in [10]. SMB is also the first ensemble solution, to the best of our knowledge, that can **choose** the appropriate matchers for the ensemble. Current state-of-the-art focuses only on setting appropriate weights to matchers that are predefined for an ensemble (see sections 4 and 5 for detailed discussions).

We present a comparative empirical results showing that SMB outperforms, on average, any individual matcher**.** While some matchers may provide, for this or that instance, a better matching then SMB, no matcher besides SMB performs constantly better. In our experiments we have compared SMB with more than 30 other matchers over a real world data of 230 schemata and several ensembling approaches, including the Meta-Learner of LSD.

The main contribution of this work is twofold. First, we introduce a new schema matching ensembling algorithm that can both choose the best matchers for an ensemble and set weights to the different matchers in a way that improves significantly on their individual performance. Second, using a thorough empirical analysis with a large data set of real-world schemata, we show a clear dominance of SMB over other existing ensemble generators.

The rest of the paper is organized as follows. Section 2 presents the schema matching model. Section 3 introduces SMB, a new boosting heuristic for schema matching and discusses several variations to tackle specific characteristics of schema matchers. We next present a comparative empirical analysis in Section 4. We conclude with an overview of related work (Section 5) and summary (Section 6).

2 Background and Model

Let *schema* $S = \{A_1, A_2, ..., A_n\}$ be a finite set of *attributes*. Attributes can be both simple and compound, compound attributes should not necessarily be disjoint, *etc.* For example, a compound attribute may be an XSD element `note` with nested elements `to`, `from`, `heading`, and `body`. For any schemata pair S and S', let $\mathcal{S} = S \times S'$ be the set of all possible *attribute matchings* between S and S'. Let $M(S, S')$ be an $n \times n'$ *similarity matrix* over $\mathcal{S}$, where $M_{i,j}$ represents a degree of similarity between the i-th attribute of S and the j-th attribute of S'. Most schema matching works define $M_{i,j}$ to be a real number in $(0, 1)$.

Let the power-set $\Sigma = 2^{\mathcal{S}}$ be the set of all possible *schema matchings* between the schema pair S and S'.[1] Formally, the input to the process of schema matching is given by two schemata S and S' and a constraint boolean function $\Gamma : \Sigma \rightarrow \{0, 1\}$, capturing the application-specific constraints on schema matchings, *e.g.*, cardinality constraints and inter-attribute matching constraints. The output of the schema matching process is a *schema matching* $\sigma \in \Sigma_\Gamma$, where $\Sigma_\Gamma = \{\sigma \in \Sigma \mid \Gamma(\sigma) = 1\}$ is the set of all *valid* schema matchings for which $\Gamma(\sigma) = 1$ (meaning that the matching σ can be accepted by a designer). Modeling Γ is beyond the scope of this work.

Each schema matching σ is associated with a schema measure of similarity $\Omega(\sigma)$, typically computed as a function of the participating attribute similarities in $M(S, S')$. For example, a typical schema measure of similarity is computed as the average of attribute pair similarity measures.

2.1 Schema Matchers

Schema matchers are instantiations of the schema matching process. They differ mainly in the measures of similarity they employ, yielding different similarity matrices. These measures can be arbitrarily complex, and may use various techniques for name matching, structure matching, *etc.*

To illustrate our model and for completeness sake we now present a few examples of schema matchers, representative of many other, similar matchers. Detailed description of these matchers can be found in [11, 17]:[2]

Term: Term matching compares labels and names to identify syntactically similar terms. To achieve better performance, terms are preprocessed using several techniques originating in IR research. Term matching is based on either complete word or string comparison. As an example, consider the terms `airline information` and `flight airline info`, which after concatenating and removing white spaces become `airlineinformation` and `flightairlineinfo`, respectively. The maximum common substring is `airlineinfo`, and the similarity of the two terms is $\frac{length(\texttt{airlineinfo})}{length(\texttt{airlineinfomation})} = \frac{11}{18} = 61\%$.

Value: Value matching utilizes domain constraints (*e.g.*, drop lists, check boxes, and radio buttons). It becomes valuable when comparing two terms that do not exactly match through their labels. For example, consider attributes `Dropoff Date` and `Return Date`. These two terms have associated value sets {(*Select*)*,1,2,...,31*} and {(*Day*)*,1,2,...,31*} respectively, and thus their content-based similarity is $\frac{31}{33} = 94\%$, which improves significantly over their term similarity ($\frac{4(\texttt{Date})}{11(\texttt{DropoffDate})} = 36\%$).

Composition: A composite term is composed of other terms (either atomic or composite). Composition can be translated into a hierarchy. This schema

[1] For ease of exposition, we constrain our presentation to a matching process of two schemata.

[2] OntoBuilder algorithm description is also available online at http://iew3.technion.ac.il/OntoBuilder/Data/10.OntoBuilder_Papers/dis.pdf

matcher assigns similarity to terms, based on the similarity of their neighbors. The Cupid matcher [15], for example, is based on term composition.

Precedence: The order in which data are provided in an interactive process is important. In particular, data given at an earlier stage may restrict the options for a later entry. For example, a car rental site may determine which car groups are available using the information given regarding the pick-up location and time. Therefore, once those entries are filled in, the information is sent back to the server and the next form is brought up. Such precedence relationships can usually be identified by the activation of a script, such as the one associated with a SUBMIT button. Precedence relationships can be translated into a precedence graph. The matching algorithm is based on a technique we dub *graph pivoting*, as follows. When matching two terms, we consider each of them to be a pivot within its own ontology, thus partitioning the graph into a subgraph of all preceding terms and all succeeding terms. By comparing preceding subgraphs and succeeding subgraphs, we determine the confidence strength of the pivot terms. Precedence was used in [28] to determine attribute correspondences with a holistic matcher.

Term and Value: A weighted combination of the Term and Value matchers. Here, the input to the matcher involves similarity matrices.

Combined: A weighted combination of the Term, Value, Composition, and Precedence matchers.

For the sake of this work, we separate matchers that are applied directly to the application (*e.g.*, Term) from matchers that are applied to the outcome of other matchers (*e.g.*, Combined). We call the former *first line matchers* and the latter *second line matchers*. The two second line matchers introduced above are based on weighted average of other matchers. We now introduce a few more second line matchers that are based on constraint satisfaction.

The Maximum Weighted Bipartite Graph (MWBG) algorithm and the Stable Marriage (SM) algorithm, both enforcing a cardinality constraint of 1:1. In [17] we have introduced a heuristic we dub Intersection that simply computes and outputs the intersection set of both algorithm outputs. For comparison sake, we also suggest Union, which includes in the output mapping any attribute mapping that is in the output of either MWBG or SM. It is worth noting that neither Intersection nor Union enforce $1:1$ matching.

A variation of the SM matcher is the Dominants matcher. The matcher chooses *dominant pairs*, those pairs in the similarity matrix with maximum value both in their row and in their column. The main assumption guiding this heuristic is that the dominant pairs are the most probable to be in the exact matching since the two attributes involve in a dominant pair prefer each other most. Note that with this heuristic not all the target attributes are mapped and that an attribute in one schema may be mapped to more than one attribute in another schema, whenever attribute pairs share the same similarity level.

Finally, in [16] we have introduced 2LNB, a second line matcher that uses a Naïve Bayes classifier over matrices to determine attribute matchings. Autoplex

[2], LSD [5], iMAP [3], and sPLMap [21] also use a naïve Bayes classifier to learn attribute mappings probabilities using instance training set.

2.2 Monotonicity

The SMB heuristic, proposed in this paper, uses similarity measures of other matchers when computing its own similarity measures. To do so, the input to the heuristic needs to be "meaningful." In particular, what if the matcher provides us with an arbitrary similarity number? In [10], monotonicity was introduced as a justification to decision making with similarity measures. In the appendix, we provide a brief description of monotonicity and statistical monotonicity for completeness sake. Intuitively, a schema matcher is statistically monotonic with respect to given two schemata if the expected similarity measure increases with precision.

3 The SMB Heuristic

Research has shown that many schema matchers perform better than random choice. In Section 2.2 and the appendix we have presented the monotonicity principle and we argue that any (statistically) monotonic matcher is a weak classifier [24]. A *weak classifier* is a classifier which is only slightly correlated with the true classification and its hypotheses are at least slightly better than random choice. The theory of weak learners has led to the introduction of a class of *Boosting* algorithms (*e.g.*, [24]). This class of algorithms can strengthen weak learners to achieve arbitrarily high accuracy and has been shown to be effective in the construction of successful classifiers. Given a set of weak classifiers, the algorithm iterates over them while re-weighting the importance of elements in the training set. There exist many versions of boosting to-date. In this paper we build upon the AdaBoost algorithm [9], described in Section 3.1 for completeness sake. AdaBoost is the most popular and historically most significant boosting algorithm. Section 3.2 then introduces our new heuristic, Schema Matcher Boosting (SMB) followed by a discussion on how to improve the training process (Section 3.3).

3.1 AdaBoost

The input to a boosting algorithm is a set of m examples where each example (x_i, y_i) is a pair of an instance x_i and the classification of the instance mapping, y_i. While not necessarily so, y_i typically accepts a binary value in $\{-1, +1\}$, where -1 stands for an incorrect classification and $+1$ stands for a correct classification. Therefore, the algorithm is aimed at binary classifications. The last input element is a hypothesis space $\mathcal{H}$, a set of weak classifiers.

The algorithm works iteratively. In each iteration the input set is examined by all weak classifiers. However, from iteration to iteration the relative weight of examples changes. The common technique in the boosting literature, which we follow here as well, is to place the most weight on the examples most often

Algorithm 1 `Boosting`

1: `Input:` $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, $x_i \epsilon \mathcal{X}$, $y_i \epsilon \{-1, +1\}$ and a space hypotheses $\mathcal{H}$.
2: /* initialization: */
3: $D_1(i) = 1/m$
4: $t = 1$
5: **repeat**
6: /* training phase: */
7: Find the classifier $h_t : \mathcal{X} \rightarrow \{-1, +1\}$, $h_t \epsilon \mathcal{H}$ that minimizes the error with respect to the distribution D_t: $h_t = \arg_{h_j} \min \varepsilon_j$.
8: **if** $\varepsilon_t \leq 0.5$ **then**
9: Choose $\alpha_t \epsilon R$. $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$
10: Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor
11: $t = t + 1$
12: **end if**
13: **until** $t = T$ or $\varepsilon_t > 0.5$
14: /* upon arrival of a new instance: */
15: Output the final classifier: $H(x) = sign(\sum_{k=1}^{\min(t,T)} \alpha_k h_k(x))$

misclassified in preceding iterations; this has the effect of forcing the weak classifiers to focus their attention on the "hardest" examples. Line 3 of the algorithm assigns an initial equal weight to all examples (see Section 3.3 for a revision of this initialization). Weights are updated later in line 10 (see below). An iteration counter t is set to 1 in Line 4 and incremented in Line 11.

Line 7 applies weak classifiers in parallel, looking for the most accurate h_t over the weighted examples. The amount of error of each weak classifier is computed. The error measure may take many forms (see discussion below) and in general should be proportional to the probability of classifying incorrectly an example, under the current weight distribution ($\Pr_{i \sim D_t} [h_t (x_i) \neq y_i]$). At round t, the weak classifier that minimizes the error measure of the current round is chosen.

Lines 8 and 13 provide a stop condition, limiting the amount of error to be no more than 50%. In addition, a restriction on the maximum number of iterations is also part of a stop condition. In Line 9, the amount of change to example weights α_t is determined. In [8], it was shown that for binary classifiers, training error can be reduced most rapidly (in a greedy way) by choosing α_t as a smoothing function over the error. Such a choice minimizes $Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$. In Line 10, the new example weights are computed for the next round $(t+1)$, using Z_t as a normalization factor.

Lines 1-13 of Algorithm 1 serve for training the algorithm weights. These weights are then used in Line 15 to classify a new instance x, by producing $H(x)$ as a weighted vote, where α_k is the weight of the classifier chosen in step k and $h_k(x)$ is the decision of the classifier of step k.

The complexity of the training phase varies, as the number of iterations is not pre-determined and the cost of applying the weak classifiers may vary. However, training is done offline and the cost of using it is linear in the size of $\mathcal{H}$.

3.2 SMB: Schema Matcher Boosting

The Boosting algorithm is trivially simple. However, Algorithm 1 is merely a shell, serving as a framework to many possible instantiations. What separates a successful instantiation from a poor one is the selection of three elements, namely the instances (x_i), the hypothesis space $\mathcal{H}$, and the error measure ε_t. We next show the SMB heuristic as a concrete instantiation of Algorithm 1, tailor-made to our specific problem domain of schema matching.

The example set $\{(x_i, y_i)\}$ consist of a set attribute pairs (x_i is a pair!), one attribute from each schema, and of the classification of the instance mapping y_i. Such a pair represents an attribute matching. This approach can be easily extended to select multiple attributes from each schema, as long as the matcher itself can assess the similarity measure of multiple attributes. Also, to support holistic matching, examples can be designed to be sets of attributes from multiple schemata rather than a pair. Each instance x_i can be correct (*i.e.*, belongs to the exact matching) or incorrect. Therefore, y_i can have two possible values $(+1)$ (for a correct matching) and (-1) (for an incorrect matching).

Choosing the hypothesis space is more tricky. Following our model in Section 2, we first note that the input to the proposed heuristic is no longer the schemata S and S', but rather a similarity matrix $M(S, S')$ (together with Γ, the constraint enforcer function). Given schemata S and S', we denote by $\mathcal{M}(S, S')$ the (possibly infinite) set of similarity matrices $M(S, S')$. The SMB heuristic is a mapping $\mathsf{SMB} : \mathcal{M}(S, S')^* \Gamma\times \rightarrow \mathcal{M}(S, S')$, transforming one (or more) similarity matrices into another similarity matrix. Therefore, we define the elements of the hypothesis space to be matrices. After experimenting with several variations, the most promising hypothesis space seems to be a set of second line matchers (as defined in Section 2), of the type decision makers (whose output is a binary matrix. For example, a hypothesis h in $\mathcal{H}$ is (Term, Dominants), where the Dominants second line matcher is applied to the outcome of the Term first line heuristic. Among other things, Dominants serves in enforcing the domain constraints, as expressed by Γ. It is worth noting SMB is also a decision maker and the outcome of SMB is a binary matrix.

Finally, we address the form of the error measure ε. A matcher can either determine a correct attribute matching to be incorrect (false negative) or it can determine an incorrect attribute matching to be correct (false positive). Let A_t denote the total weight of the false negative examples, C_t denotes the total weight of the false positive examples, and B_t denotes the total weight of the true positive examples, all in round t. Typically, one would measure error in schema matching in terms of precision and recall, translated into boosting terminology as follows:

$$P(t) = \frac{B_t}{C_t + B_t}; R(t) = \frac{B_t}{A_t + B_t} \tag{1}$$

Precision and recall may be combined in many ways, one of which is the F-Measure, their harmonic mean:

$$F(t) = \frac{2B_t}{A_t + C_t + 2B_t} \tag{2}$$

and therefore, a plausible error measure for the SMB heuristic is:

$$\varepsilon_t = 1 - F(t) = 1 - \frac{2B_t}{A_t + C_t + 2B_t} = \frac{A_t + C_t}{A_t + C_t + 2B_t} \tag{3}$$

This is indeed the measure we present in this paper. It is worth noting, however, that this is not the only measure possible. Our empirical evaluation (not shown in this work) suggests that Eq. 3 performs better than other error measures.

Example 1. The example is taken from one of our experiments, described in Section 4. Given the hypotheses space $\mathcal{H}$ as described above, and given a dataset of size 70, the SMB heuristic performs 5 iterations: It started by creating a dataset with equal weight for each mapping. In the first iteration, it picked (Composition, Dominants)[3] as the weak hypothesis which was the most accurate hypothesis over the initial weight distribution ($\varepsilon_1 = 0.328 \Rightarrow \alpha_1 = 0.359$). In the second iteration, the selected hypothesis was (Precedence, Intersection) with $\varepsilon_2 = 0.411$ and $\alpha_2 = 0.180$. In the third iteration, the selected hypothesis was (Precedence, MWBG) ($\varepsilon_3 = 0.42 \Rightarrow \alpha_3 = 0.161$). The fourth hypothesis involved (Term and Value, Intersection) with error $\varepsilon_4 = 0.46$ and $\alpha_4 = 0.080$. The final hypothesis was for (Term and Value, MWBG) with error $\varepsilon_5 = 0.49 \Rightarrow \alpha_5 = 0.020$. In the sixth iteration no hypothesis had accuracy less than 50% so the training phase is terminated having 5 iterations each one with its strength α_t. The outcome classification rule is a linear combination of the five weak hypothesis with their strength as coefficients. So, given a new mapping (a, a') to be classified, each one of the weak classifiers contributes to the final decision by its weak decision weighted by its strength and if the final decision was positive then the given mapping would be classified as a correct mapping. Otherwise, it would be classified as an incorrect one.

Let $\tilde{h}_{\max}$ be the maximum execution time of a matcher in $\mathcal{H}$ and $t_{\max}$ be the number of iteration performed by SMB. The training time of SMB is $O\left(\tilde{h}_{\max} \cdot t_{\max}\right)$. Given a new schema pair, let $n_{\max}$ be the maximum number of attributes in each schema. The cost of using SMB is $O\left(n_{\max}^2\right)$, the cost of generating the output matrix.

Two comments avout α_t: First, our choice of α_t limits it to be non-negative, since ε_t is restricted not to exceed 0.5 (see lines 8 and 9 of Algorithm 1). This is one characteristic that differentiates SMB from the Meta-Learner of LSD, which uses a least-square linear regression on the training data set. We shall elaborate on this difference more in Section 4.2. Secondly, if a hypothesis is chosen more than once during the training phase, its total weight in the decision making process is the sum of all the weights α_t with which it has been assigned.

3.3 Preprocessing the training dataset

Tracing the evolution of the error computed in each iteration of SMB, we observe that error increases very quickly. Recall that the error is the sum of all the

[3] Description of all matchers in this example are given in Section 2.1.

weights of the incorrectly classified examples. Therefore, we hypothesize that this phenomenon is a result of outliers, *i.e.*, examples that are inherently ambiguous and hard to categorize. Therefore, **no** matcher in the ensemble classifies them correctly. Even with a small number of outliers, the emphasis placed on the incorrectly classified examples becomes detrimental to the performance of SMB. Such examples receive increasing weights with each new iteration. Consequently, error accumulation accelerates rather than subsides. Regardless of which classifier is chosen, such examples will be misclassified and their weights will increase.

To avoid this phenomenon, we introduce a preprocessing phase to the training phase of SMB, in which we identify and filter outliers. These examples are de-emphasized by eliminating them from the training set to avoid rapid error accumulation. We remove all examples which no matcher classifies correctly. Eliminating examples from the training set is equivalent to a presetting small weights to outliers. Our empirical analysis has shown that the preprocessing stage yields a significant improvement in performance.

4 Experiments

4.1 Experiment setup, data, and evaluation methodology

Experiment setup In our experiments we have used 30 matcher combinations (recall that our hypothesis space is made of matching pairs), combining Term, Value, Composition, Precedence, Term and Value, and Combined with MWBG, SM, Dominants, Intersection, Union, and 2LNB. All matchers were described in Section 2.1. All algorithms were implemented using Java 2 JDK version 1.5.0_09 environment, using an API to access OntoBuilder's matchers and get the output matrices. The experiments were run on a laptop with Intel Centrino Pentium m, 1.50GHz CPU, 760MB of RAM Windows XP Home edition OS.

The Term and Combined matchers were shown in [10] to be monotonic. Our preliminary experiments show that Value was not monotonic, and is brought here as a baseline case. To demonstrate the potency of our matchers, we have experimented with the OAEI 2006 Directory benchmark.[4] Our empirical analysis yields that the pair (Term, MWBG), for example, achieved on average Precision of 61%, Recall of 96%, and F-Measure of 72%, on a set of 110 randomly selected tasks. This is better than other known results on this data set.

Data set For our experiments, we have selected 230 Web forms from different domains, such as job hunting, Web mail, and hotel reservation. We extracted a schema from each Web form using OntoBuilder. We have matched the Web forms in pairs (115 pairs), where pairs were taken from the same domain, and generated manually the exact matching for each pair.[5] The schemata vary in size, from 8

[4] The benchmark is publicly available at http://keg.cs.tsinghua.edu.cn/project/RiMOM/oaei2006/oaei2006.html. Our results are based on a private evaluation of the exact matching, since the OAEI organizers do not provide the exact matching.

[5] All ontologies and exact matchings are available for download from the OntoBuilder Web site, http://ie.technion.ac.il/OntoBuilder.

to 116 attributes with about two thirds of the schemata have between 20 and 50 attributes. They also vary in the proportion of number of attribute pairs in the exact matching relative to the target schema.[6] This proportion ranges from 12.5% to 100%; the proportion in about half of the ontologies is more than 70%, which means that about 70% of the schema attributes can be matched. Another dimension is the size difference between matched schemata, ranging from equal size schemata to about 2.2 times difference between schemata. In about half of the pairs, the difference was less than 50% of the target schema size.

We ran the six schema matchers (Term, Value, Composition, Precedence, Term and Value, and Combined) on the 115 pairs, generating 690 matrices. These matrices used the second line matchers (MWBG, SM, Dominants, Intersection, Union, and 2LNB) to generate new matrices. 2LNB was paired only with the Combined matcher. All in all, we have analyzed 3565 pairs of real-world schemata.

Evaluation methodology We have repeated experiments with a varying size of training dataset. Here, we report on experiments with a training set of 60 randomly selected schema pairs and a test set of size 30 (schema pairs) was also selected randomly from the remaining matrices. We have repeated each experiment three times. Preprocessing to avoid outliers was applied.

To evaluate the various heuristics, we use Precision and Recall. Lower precision means more false positives, while lower recall suggests more false negatives. To extend Precision and Recall to the case of non 1 : 1 mappings, we have adopted a correctness criteria according to which any attribute pair that belongs to the exact mapping is considered to be correct, even if the complex mapping is not fully captured.

4.2 Results and analysis

Comparative Performance Analysis We first analyze comparatively the performance of SMB with the 31 matcher pairs. Figure 1(left) position all 32 matchers on a Precision (x-axis) vs. Recall (y-axis) scatter plot. The expected Precision/Recall tradeoff is evident here, with no single dominating matcher. Matchers are partitioned into two groups. To the right there are all those matchers that qualify as weak classifiers, which we define to be those whose F-Measure is higher than 50%. To the left, there are 5 matchers that cannot be considered weak matchers. In common to all 5 matcher pairs is the use of the Value matcher. This matcher is not statistically monotonic, since it cannot differentiate between pairs that share the same attribute domain. However, when combined with the Term matcher, Value generally adds 1-2% to the Term matcher performance.

SMB is clearly the winner in terms of Precision, balancing Precision with Recall. Figure 1(right) illustrates the percentage of improvement SMB provides in terms of Precision. The x-axis represents the different matcher pairs while the y-axis shows the percentage of improvement. It ranges from 5.4% to 66% for weak matchers. For example, SMB improved by 39% over the pair (Term, MWBG), illustrated earlier to have good outcomes on tough data sets (such

[6] In OntoBuilder, one of the schemata is always chosen to be the target schema, the schema against which comparison is performed.

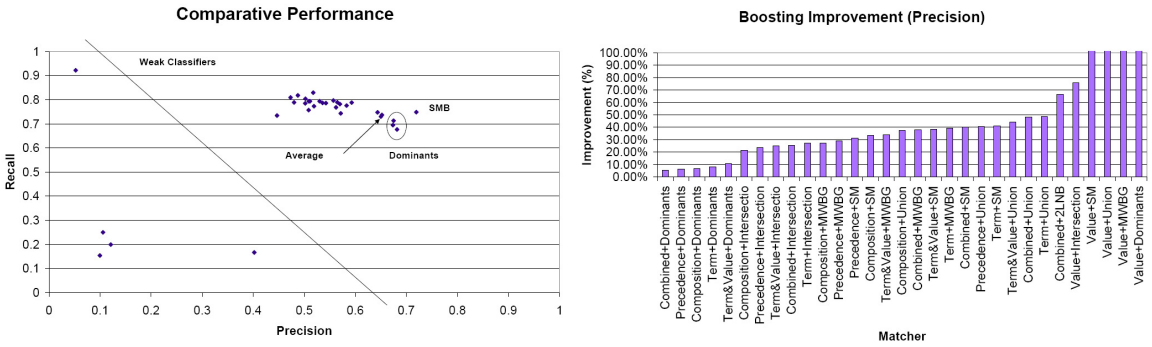

Fig. 1. Performance Analysis

as the directory dataset of OAEI'2006). In terms of F-Measure, we observe an improvement of 4.3-34.3% for weak matchers.

For comparison, Figure 1(left) also contains another matcher weighing technique, Average, which is discussed in details later in this section. While Average blends in with other matchers (mainly from the (*, Dominants) set), SMB stands out in its Precision performance, while not compromising much its Recall.

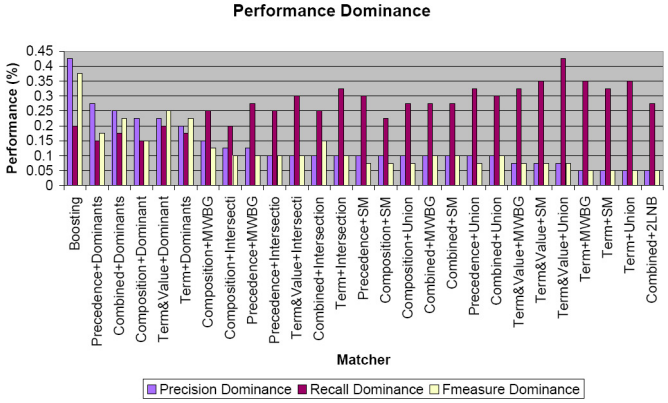

Fig. 2. Dominance Analysis

Such an improvement is nice, yet not unheard of. For example, LSD has shown an improvement of 5-22% in Precision [5]. We defer a comparison with LSD to later in the section, arguing here that these results are comparable. while SMB may be better than the other individual matchers on average, how often does it manage to outperform all other matchers? we have analyzed the data and the results are illustrated in Figure 2. For each matcher, we record the

percentage of schema pairs, where it was not outperformed by any of the other matchers in terms of Precision, Recall, and F-Measure. The figure provides a comparison of all weak classifiers and SMB. SMB clearly performs the best in terms of Precision and F-Measure. In 43% of the schema pairs, its Precision performance was not dominated by any other matcher. The next best matcher in this category was (Precedence, Dominants), non dominated in only 28% of the cases.[7] Similar results are observed for the F-Measure, where SMB leads with 38%, followed by (Term&Value, Dominants) with 25%. For Recall, SMB is non-dominated for 20% of the schema matching pairs.

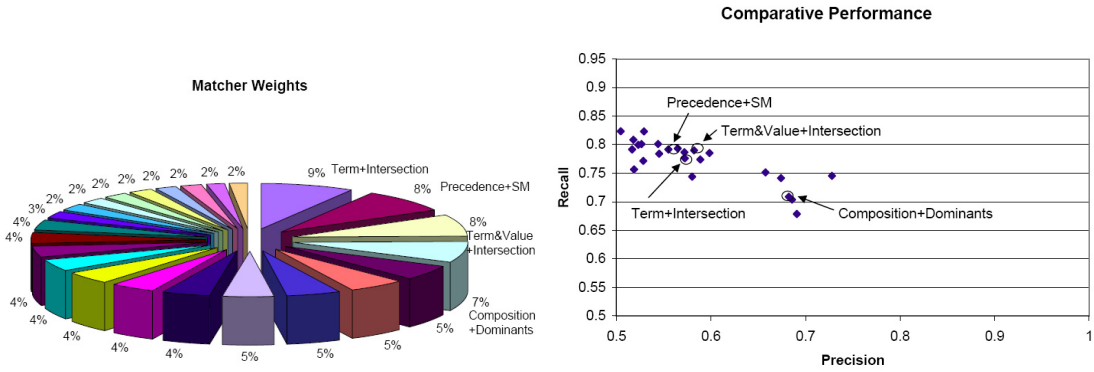

Fig. 3. Relative matcher weights in SMB and individual performance

Matcher Selection We now analyze the decision making process of SMB. Given the individual performance of each matcher, one could expect that those matchers with the highest weight in the decision making of SMB will be those that perform best individually. In our case, the top 4 matchers, in terms of Precision and F-Measure are pairs in which the second line matcher is Dominants. Figure 3 presents the relative matcher weights in SMB. The higher the weight, the more important is the vote of a matcher regarding each attribute matching. Only 24 out of the 31 matchers participate in the decision making of SMB.

Surprisingly, the top 4 matchers in Figure 3(left) include only one pair with Dominants ((Composition, Dominants)). The pair (Combined, Dominants) is the leading pair in terms of Precision when observing individual performance, yet is not even part of the SMB decision making (!). The most important matcher for SMB is (Term, Intersection), ranked 11-th according to the F-Measure individual performance and 10-th according to Precision. (Precedence, SM), ranked second for SMB, has a mediocre individual performance. Figure 3(right) is a zoomed in version of Figure 1(left), highlighting the four top matchers of SMB.

[7] Note that the sum of non-dominance percentage exceeds 100% since matchers may reach the same level of precision for some instances, counting both to be non-dominated for this instance.

Weight Scheme	Avg. Precision	Avg. Recall	Avg. F-Measure
Boosting	0.73	0.75	0.74
Meta-Learner	0.57	0.69	0.62
F-Measure	0.61	0.72	0.64
Average	0.65	0.73	0.67
Random	0.57	0.65	0.61

Table 1. Comparison of weight schemes

Our first observation is that the decision making of SMB is not linear in the individual performance of matchers, and therefore the training process of SMB is valuable. Secondly, we observe that SMB seeks diversity in its decision making. It uses Term, Value (combined with Term due to its individual poor performance), Composition, and Precedence. Given these four matchers, SMB has no need for the Combined matcher, which provides a weighted average of the four. This explains the absence of (Combined, Dominants).

A surprising property for SMB is its ability to **choose** matchers for an ensemble. Lee et al. suggested in [13] that the tuning of an ensemble involves the selection of "the right component to be executed." However, to the best of our knowledge, none of the existing algorithmic solutions offer such a selection feature. eTuner suggests a method for tuning "knobs" given an ensemble but does not provide a method for constructing it. LSD also applies the Meta-Learner to an existing ensemble. SMB performs its tuning sequentially. It starts by greedily choosing those matchers that provides a correct solution to a major part of the schema matching problem. Then, it adds matchers that can provide insights to solving the harder problems. Those matchers that are left out will not be part of the ensemble. In our experiments, SMB includes only 24 out of the 31 matchers. We consider this feature as a main contribution of the proposed algorithm.

Weight Selection The outcome of SMB matcher training is a weighted average for matcher voting. Our next set of experiments, summarized in Table 1, compare the outcome of using SMB weights with other weighing schemes.

The first row represents the average performance of SMB, as presented above (Figure 1). In the second row, the performance of the LSD's Meta-Learner is presented. Given a set of weak learners $\mathcal{H}$, the Meta-Learner uses a least-square linear regression on the training data set, minimizing the squared error

$$\sum_{i=1}^{m} \left(y_i - \sum_{h \in \mathcal{H}} h\left(x_i\right) \cdot w_h \right)^2 \tag{4}$$

where y_i is set to 0 if the pair x_i should not be matched and 1 otherwise. $h\left(x_i\right)$ is the decision of learner h regarding pair x_i and w_h are the variables on which the linear regression is applied. The Meta-Learner cannot choose classifiers and therefore, with 31 different learners, a huge space of possibilities exist. To allow a comparison using some common baseline, we have selected the top 16

matchers chosen by SMB to participate in the training of the Meta-Learner. The remaining 8 matchers seem to have little impact on the decision making of SMB. The most dominant learner was (Combined, Intersection), which was ranked 8-th by SMB, demonstrating that the Meta-Learner and SMB reach different decisions regarding matcher importance. For example, the learner (Term, Dominants) (ranked 9-th by SMB) has received a negative weight. It is worth noting that unlike SMB, the weights of the Meta-Learner can be negative as well. This has the interesting effect of transposing the decision of a matcher.

Once the weights have been set, a set of 30 schema pairs was chosen randomly and generated the outcome using the weights w_h from the training phase. The Meta-Learner reached a precision of 57%, a recall of 69%, and an F-Measure of 62%. Comparing with the results of SMB, we observe that SMB performs 28% better than the Meta-Learner in terms of precision, 9% better in terms of recall and 19% better in terms of F-Measure.

In the third line we present the results of matching 30 randomly chosen schema pairs, where matchers are assigned a weight equivalent to their F-Measure. For example, (Precedence, Dominants) is assigned a weight of 0.69 while (Value, Dominants) is assigned a weight of 0.11. This weighing scheme reduces Precision by about 20% on average, Recall by about 4% on average, and F-Measure by about 16% on average. The performance of weighing using F-Measure is worse (!) than those of assigning equal weights to the various matchers (fourth line of Table 1; also presented in Figure 1(left)). SMB improves precision by 12%, recall by 3%, and F-Measure by 11%.

The fifth row provides the average result of 3 random weight selections, each time testing the random weight over 30 randomly selected schema pairs. SMB improves precision by 28%, recall by 15%, and F-Measure by 21%.

To conclude, in this set of experiments, SMB is shown to dominate all other tested weighing scheme, in terms of Precision, Recall, and F-Measure. This, together with the ability of SMB to select matchers for an ensemble, sums up to show SMB to be the best choice for ensemble design.

5 Related Work

25 years of schema matching research, first as part of schema integration and then as a standalone research, are summarized in surveys [1, 22, 27] and various online lists, *e.g.*, OntologyMatching,[8] Ziegler,[9] DigiCULT,[10] and SWgr.[11]

Machine learning has been used for schema matching in several works. Autoplex [2] and LSD [5] use a Naïve Bayes classifier to learn attribute mappings probabilities using instance training set. SEMINT [14] use neural networks to

[8] http://www.ontologymatching.org/
[9] http://www.ifi.unizh.ch/~pziegler/IntegrationProjects.html
[10] http://www.digicult.info/pages/resources.php?t=10
[11] http://www.semanticweb.gr/modules.php?name=News&file=categories&op=newindex&catid=17

identify attribute mappings. APFEL [6] determines heuristic weights in an ensemble and threshold levels using various machine learning techniques, namely decision trees (*e.g.*, C4.5), neural networks, and support vector machines. C4.5 was also used in [29], using WordNet relationships as features. sPLMap [21] use Naïve Bayes, kNN, and KL-distance as content-based classifiers. All these works applied machine learning directly to the schemata, while our approach uses the outcome of other matchers for learning and improvement. In particular, the use of boosting was never applied to schema matching, to the best of our knowledge.

Research into ensemble design include eTuner [13], LSD [5] and others (*e.g.*, [16]). SMB is similar to the Meta-Learner in [5]. In both approaches a set of matchers is selected and a weighted average of the decisions taken by these matchers determine the matching outcome. In [5], the weights where set using a least-square linear regression analysis while we use the boosting mechanism. The literature shows the connection between boosting and logistic regression [26], yet there is no evident connection to linear regression. Our empirical results show that SMB outperforms the Meta-Learner.

6 Conclusions

In this work we have presented the Schema Matcher Boosting (SMB) heuristic to efficiently use an ensemble of matchers. We have analyzed, both conceptually and empirically, the properties of SMB, discussing its benefit and analyzing its performance. SMB has the unique ability to choose from a pool of matchers. Its decision making is based on diversity of matchers, taking their best combination. Our empirical analysis also shows that SMB provides a major increase in Precision with no or minimal loss of Recall. SMB performance improves on any individual matcher pair with which we have experimented and was shown to dominate other weighing schemes, including that of LSD Meta-Learner.

In our future work, we aim at improving SMB even more. We intend to look at existing works in boosting, involving multiclass classification (*e.g.*, AdaBoost.M1 and AdaBoost.M2, [8]) and error-correcting output codes [25], possibly identifying new ties to the schema matching problem. Another direction will be to incorporate human knowledge, as was suggested by several schema matching papers in the past. We shall look into works such as [23], where human judges construct estimated probability. This approach was argued to be too hard for expert to deal with in schema matching, so we shall look into indirect methods for building such estimated probability functions.

References

1. C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, Dec. 1986.
2. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento,*

Italy, September 5-7, 2001, Proceedings, volume 2172 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001.
3. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex mappings between database schemas. In *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 383–394, 2004.
4. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the International conference on Very Large Data Bases (VLDB)*, pages 610–621, 2002.
5. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In W. G. Aref, editor, *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 509–520, Santa Barbara, California, May 2001. ACM Press.
6. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with apfel. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, pages 186–200, 2005.
7. D. Embley, D. Jackman, and L. Xu. Attribute match discovery in information integration: Exploiting multiple facets of metadata. *Journal of Brazilian Computing Society*, 8(2):32–43, 2002.
8. Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 1997.
9. Y. Freund and R. Schapire. A short introduction to boosting, 1999.
10. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 14(1):50–67, 2005.
11. A. Gal, G. Modica, H. Jamil, and A. Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 26(1):21–32, 2005.
12. R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 51–61. ACM Press, 1997.
13. Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. *VLDB Journal*, 16(1):97–122, 2007.
14. W.-S. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000.
15. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International conference on Very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, Sept. 2001.
16. A. Marie and A. Gal. Managing uncertainty in schema matcher ensembles. In H. Prade and V. Subrahmanian, editors, *Scalable Uncertainty Management, First International Conference, SUM 2007*, pages 60–73, Washington, DC, USA, Oct. 2007. Springer.
17. A. Marie and A. Gal. On the stable marriage of maximumweight royal couples. In *Proceedings of AAAI Workshop on Information Integration on the Web (II-Web'07)*, Vancouver, BC, Canada, July 2007.
18. S. Melnik. *Generic Model Management: Concepts and Algorithms*. Springer-Verlag, 2004.
19. S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 193–204, San Diego, California, 2003. ACM Press.

20. P. Mork, A. Rosenthal, L. Seligman, J. Korb, and K. Samuel. Integration workbench: Integrating schema integration tools. In *Proceedings of the 22nd International Conference on Data Engineering Workshops, ICDE 2006, 3-7 April 2006, Atlanta, GA, USA*, page 3, 2006.
21. H. Nottelmann and U. Straccia. Information retrieval and machine learning for probabilistic schema matching. *Information Processing and Management*, 43(3):552–576, 2007.
22. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
23. G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. In *Proceedings of the International Workshop on AI and Statistics*, page 152161, 1999.
24. R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.
25. R. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, page 313321, 1997.
26. R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, Mar. 2001.
27. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal of Data Semantics*, 4:146 – 171, Dec. 2005.
28. W. Su. *Domain-based Data Integration for Web Databases*. PhD thesis, Dept. of Computer Science and Engineering, Hong Kong Univ. of Science and Technology, Hong Kong, Dec. 2007.
29. L. Xu and D. Embley. A composite approach to automating direct and indirect schema mappings. *Information Systems*, 31(8):697–886, Dec. 2006.

Monotonicity

The evaluation of schema matchings is performed with respect to an *exact matching*, based on expert opinions. *Precision* and *recall* are used for the empirical evaluation of performance. Assume that out of the $n \times n'$ attribute matchings, there are $c \leq n \times n'$ correct attribute matchings, with respect to the exact matching. Also, let $t \leq c$ be the number of matchings, out of the correct matchings, that were chosen by the matching algorithm and $f \leq n \times n' - c$ be the number of incorrect such attribute matchings. Then, precision is computed to be $\frac{t}{t+f}$ and recall is computed as $\frac{t}{c}$. Clearly, higher values of both precision and recall are desired. From now on, we shall focus on the precision measure, denoting by $p(\sigma)$ the precision of a schema matching σ.

We first create equivalence schema matching classes on $2^{\mathcal{S}}$. Two matchings σ' and σ'' belong to a class p if $p(\sigma') = p(\sigma'') = p$, where $p \in [0, 1]$. For each two matchings σ' and σ'', such that $p(\sigma') < p(\sigma'')$, we can compute their schema matching level of certainty, $\Omega(\sigma')$ and $\Omega(\sigma'')$. We say that a matching algorithm is *monotonic* if for any two such matchings $p(\sigma') < p(\sigma'') \rightarrow \Omega(\sigma') < \Omega(\sigma'')$. Intuitively, a matching algorithm is monotonic if it ranks all possible schema matchings according to their precision level.

A monotonic matching algorithm easily identifies the exact matching. Let σ^* be the exact matching, then $p(\sigma^*) = 1$. For any other matching σ', $p(\sigma') < p(\sigma^*)$.

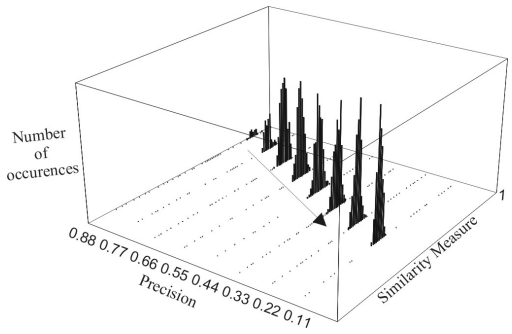

Fig. 4. Illustration of the monotonicity principle

Therefore, if $p(\sigma') < p(\sigma^*)$ then from monotonicity $\Omega(\sigma') < \Omega(\sigma^*)$. All one has to do then is to devise a method for finding a matching σ^* that maximizes Ω.[12]

Figure 4 provides an illustration of the monotonicity principle using a matching of a simplified version of the Web forms in "Absolute Agency" with "Adult Singles" Web sites, both taken from the dating and matchmaking domain. Both schemata have nine attributes, all of which are matched under the exact matching. Given a set of matchings, each value on the x-axis represents a class of schema matchings with a different precision. The z-axis represents the similarity measure. Finally, the y-axis stands for the number of schema matchings from a given precision class and with a given similarity measure.

Figure 4 provides two main insights. First, the similarity measures of matchings within each schema matching class form a "bell" shape, centered around a specific similarity measure. Such a behavior indicates a certain level of robustness of a schema matcher, assigning close similarity measures to matchings within each class. Second, the "tails" of the bell shapes overlap. Therefore, a schema matching from a class of a lower precision may receive a higher similarity measure than a matching from a class of a higher precision. This, of course, contradicts the monotonicity definition. However, the first observation serves as a motivation for a definition of a statistical monotonicity, first introduced in [10]:

Definition 1 (Statistical monotonicity). *Let $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_m\}$ be a set of matchings over schemata S_1 and S_2 with n_1 and n_2 attributes, respectively, and define $n = \max(n_1, n_2)$. Let $\Sigma_1, \Sigma_2, ..., \Sigma_{n+1}$ be subsets of Σ such that for all $1 \leq i \leq n+1$, $\sigma \in \Sigma_i$ iff $\frac{i-1}{n} \leq p(\sigma) < \frac{i}{n}$. We define M_i to be a random variable, representing the similarity measure of a randomly chosen matching from Σ_i. Σ is* statistically monotonic *if the following inequality holds for any $1 \leq i < j \leq n+1$:*

$$\bar{\Omega}(M_i) < \bar{\Omega}(M_j) \tag{5}$$

where $\bar{\Omega}(M)$ stands for the expected value of M.

[12] In [10], where the monotonicity principle was originally introduced, it was shown that while such a method works well for fuzzy aggregators (*e.g.*, weighted average) it does not work for t-norms such as min.