Learning-Based Approaches for Matching Web Data Entities

Entity matching is a key task for data integration and especially challenging for Web data. Effective entity matching typically requires combining several match techniques and finding suitable configuration parameters, such as similarity thresholds. The authors investigate to what degree machine learning helps semi-automatically determine suitable match strategies with a limited amount of manual training effort. They use a new framework, Fever, to evaluate several learning-based approaches for matching different sets of Web data entities. In particular, they study different approaches for training-data selection and how much training is needed to find effective combined match strategies and configurations.

Hanna Köpcke, Andreas Thor, and Erhard Rahm
University of Leipzig

Entity matching (also referred to as *object matching*, *entity resolution*, or *fuzzy join*) is a fundamental problem for data management and integration, in particular. It requires identifying entities referring to the same real-world object. Such entities might reside in distributed, typically heterogeneous data sources or in a single data source, such as a search engine store. Furthermore, applications and users can dynamically request entities to be matched from Web sources — for instance, via keyword searches. Entity matching's significance and difficulty has triggered a huge amount of research on the issue, and researchers have proposed numerous approaches to solving the problem, especially for structured entities in databases.[1–4]

Due to the large variety of data sources and entities, no single "best" solution exists for entity matching. Rather, we must use several match techniques to determine entities' similarity according to different criteria (product name and manufacturer similarity, for instance) and combine the individual similarity results. Determining an effective match strategy thus entails selecting individual match approaches (or *matchers*), specifying their parameters (for example, similarity thresholds), and choosing a matcher combination. Several research frameworks as well as commercial offerings help define such match strategies.[3] However, these systems almost exclusively use struc-

 1089-7801/10/$26.00

tured data sets, not heterogeneous Web data from different sources (one exception deals with matching product entities[5]). Furthermore, current frameworks are highly complex to use and tune for challenging match tasks because the typically huge number of possible matcher combinations and configurations makes it difficult and time-consuming even for domain experts to find a good match strategy.

We can use machine learning approaches such as decision trees or support vector machines (SVMs) to automatically determine ("learn") suitable matcher combinations[6,7] and thus potentially achieve a reduced tuning effort compared to manually specified match strategies. However, these learning-based approaches depend on suitable training data, and labeling training examples can incur a substantial manual effort for domain experts. Unfortunately, most published evaluation results don't disclose the size and selection of training data leaving open the degree of manual effort invested.[3]

In this study, we use a new evaluation framework, Fever (*F*ramework for *Ev*aluating *E*ntity *R*esolution), to investigate the effectiveness and training effort of learning-based methods to semi-automatically determine suitable match strategies for challenging Web data match tasks from different domains. We compare the results with manually specified and tuned match strategies for a commercial entity match implementation representing the current state of the art. For the learning-based approaches, we study two methods to select suitable training data and analyze how much training is needed to find an effective match strategy.

Matching Web Data Sources

Entities from Web data sources are particularly challenging to match because they are often highly heterogeneous with limited data quality regarding, for example, consistency among descriptions. Figure 1 illustrates some problems for the popular entity search engine Google Product Search, with duplicate entries in its search result for a specific camcorder. The entries refer to different shops that use heterogeneous names, descriptions, and other attributes for the same product and might also contain misspellings and other errors. For example, the names for the considered product "Canon Vixia HF S10" contain additional information that might complicate entity matching,

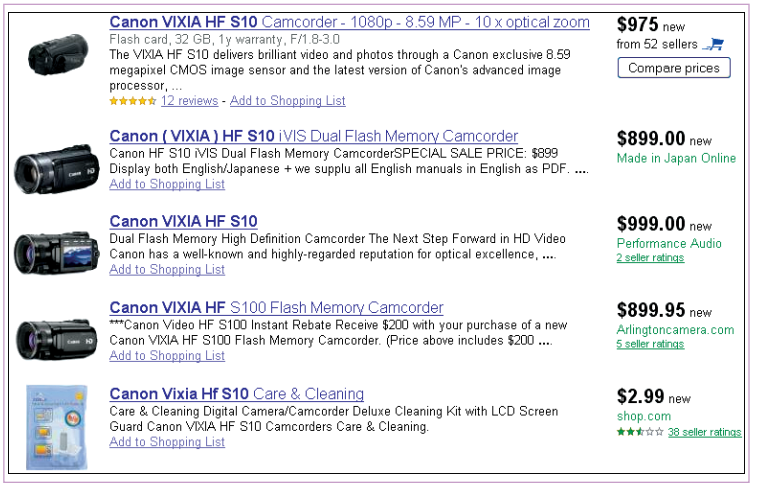

Figure 1. Duplicate Web entities in Google Product Search. The heterogeneous names, descriptions, and other attributes for the Canon Vixia HF S10 make it difficult to determine that the first three entries refer to the same product, whereas the last two don't.

making it difficult to automatically determine that the first three entries refer to the same product. On the other hand, this information can help us recognize that the fourth entry is a similar but different product, and the last entry doesn't represent the camcorder of interest, but only accessories.

Although Google clusters and ranks related products, it doesn't support sufficient entity matching. So, applications such as price comparisons would need an additional entity-matching functionality. Many domains need similar entity-matching capabilities to integrate related data entities from independent Web sources or process search results from other entity search engines (aggregate duplicate publications in Google Scholar for citation analysis, for example).

Configuring Match Strategies with Fever

We use the Fever platform[8] to evaluate several match strategies for many configurations and different match tasks. Fever supports a large spectrum of matchers and builds on our previous prototypes, Moma (*Ma*pping-based *O*bject *Ma*tching)[9] and Stem (Self-*T*uning *E*ntity *M*atching)[10] for combining several matchers. Furthermore, Fever supports different methods (operators) for blocking and training selection (which we detail later) for use within a match strategy. We define match strategies via so-called *operator trees*, specifying the operators and their execution order.

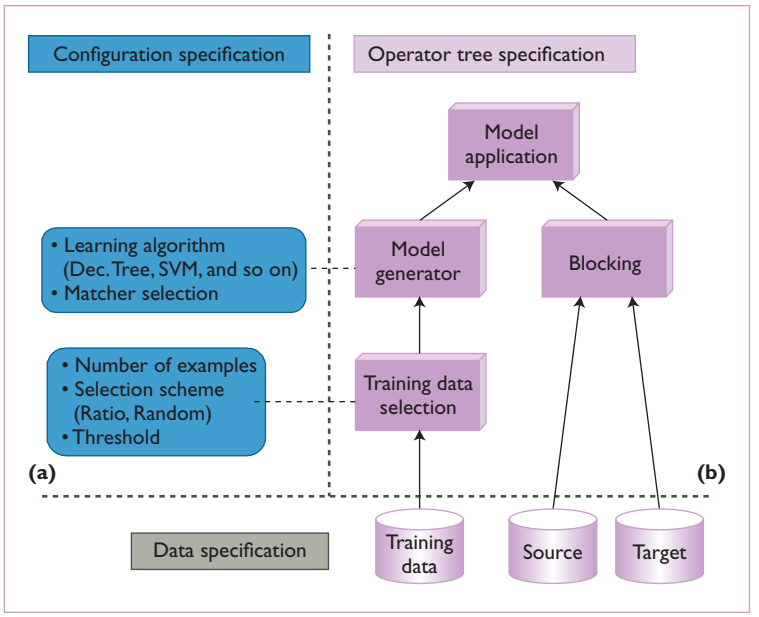

Figure 2. Fever match workflow for evaluating learning-based matchers. We can see (a) the relevant configuration specification for (b) the operator tree.

Fever represents match results as so-called *instance mappings*. A mapping m between two entity sets A and B consists of a set of match correspondences — that is, $m = \{(a, b, s) | a \in A, b \in B, s \in [0,1]\}$. The similarity value s indicates the strength of the similarity between two entities $a \in A$ and $b \in B$; we consider entity pairs with a similarity value higher than a predetermined threshold as matching. The uniform mapping data structure is the foundation for the flexible combination of operators within trees. Fever's main operator types (such as blocking, matching, and training selection) require mappings as input and generate them as output.

Fever supports two kinds of match strategies: learning-based approaches and nonlearning approaches. For nonlearning approaches, Fever provides several methods that let the system automatically evaluate numerous parameter settings for test data — for instance, a sample from the match task.[8] The user can manually inspect and compare the corresponding match results and finally select and apply the most promising configuration to the complete input data sets. In the following description, we focus on learning-based strategies, including two methods for training data selection that we study in our evaluation.

Figure 2 shows the Fever match workflow we applied in our study of learning-based matching. Figure 2b shows the operator tree, and Figure 2a shows its relevant operator parameters. Fever executes this tree in a post-order traversal sequence thereby inputting the child operators' results into the father operator.

The execution falls into two phases: *model generation* and *model application*. Model generation (left part of the operator tree) requires a training mapping that contains manually labeled correspondences representing matching (the similarity value equals 1) and nonmatching (0) entity pairs. The learning algorithm applies the specified matchers to the entity pairs in the training mapping. The learner then uses the resulting similarity values to automatically determine a match strategy model — that is, combine the specified matchers to derive a match decision for any entity pair. We discuss training selection and model generation in more detail later.

The second phase (right part of the operator tree) applies the determined model for the real match task (model application) to match a source and target data set (or to find duplicates within one data set). For efficiency reasons, exhaustively evaluating the Cartesian product of all input entities generally isn't feasible. Hence, Fever can first execute a *blocking* operator to reduce the search space to the most likely matching entity pairs. Fever supports several blocking approaches, such as sorted neighborhood or canopy clustering. For our evaluation, we use a fixed blocking strategy for all experiments — that is, blocking isn't subject to our evaluation.

Machine learning approaches' effectiveness depends on the provision of sufficient, suitable, and balanced training data. On the other hand, the number of entity pairs users must label affects the manual tuning effort and should thus be small. To address these issues, we evaluate different training sizes as well as two methods for training selection: *random* and *ratio*. Both strategies consider only entity pairs for labeling, for which the similarity exceeds a specified threshold. This ensures that the training isn't dominated by trivial nonmatching entity pairs that aren't useful for finding effective matcher parameters and combinations.

The random strategy randomly selects the specified number of entity pairs from the input exceeding the similarity threshold. The ratio method is an extension of random that aims at a certain ratio of matching and nonmatching entity pairs in the training data. It uses a

ratio parameter from the range 0 to 0.5, indicating the minimal percentage of both matching and nonmatching pairs. The ratio 0 corresponds to the random strategy enforcing no restrictions on the share of matching or nonmatching pairs. For ratio values greater than 0, Fever reduces the number of randomly selected entity pairs so that either the number of matching or nonmatching entity pairs satisfies the ratio restriction. For example, a ratio of 0.4 guarantees that at least 40 percent of all training pairs are either matching or nonmatching — in other words, at most 60 percent are nonmatching or matching. By ensuring a minimum number of pairs, the ratio approach aims to enhance the training data's discriminative value for learning effective match strategies.

For model generation, we apply a preselected set of matchers to the training data. By comparing similarity values the matchers have computed to the perfect (labeled) match result in the training, Fever can determine (learn) a combination of the most effective matchers and their parameters, such as similarity thresholds. Fever currently supports four approaches for this training-based learning of match strategies that we study in our evaluation. Three are well-known learning methods — namely, decision trees, logistic regression, and SVMs.[11] A decision tree specifies the matchers to be applied and their execution order. Each inner node of the tree tests whether a certain similarity threshold is exceeded for a specific matcher; the leaf nodes contain the match decisions. Logistic regression and SVMs determine a weighted combination of the individual matchers' similarity values. For our study, we use the open source learner implementations from RapidMiner (formerly Yale).[12] The fourth strategy is a multiple learning approach that derives its match decisions from the three basic learners' majority consensus (two entities match if at least two of the three learners vote for the match). The motivation for combined learning is to compensate for individual learners' weaknesses and thus improve overall match quality and robustness. This comes with the highest execution cost because Fever must execute the match strategies the three basic learners have determined before it can combine their results.

Fever provides many matcher implementations for use in combined match strategies. In this study, we focus on attribute matching between corresponding attributes in the input sources (product names, publication titles, and so on). We also consider four string similarity measures (Cosine, Jaccard, Term Frequency-Inverse Document Frequency [TF-IDF], and Trigram[2]) to compute string attribute values' similarity. For numerical attribute values such as product prices, we use a numerical similarity measure. Fever also supports externally implemented matchers within its operator trees. We use a commercial entity match implementation for comparison with the learning-based approaches and utilize Fever to find suitable parameter settings.

Experimental Evaluation

Let's now examine how effective learning-based match strategies can solve different match tasks on heterogeneous Web data entities in comparison to manually tuned strategies with a state-of-the-art match approach.

Evaluation Setting

We evaluate our approach for four match tasks from two application domains (bibliographic and e-commerce data entities). Table 1 provides some statistics on these tasks, which are named after the involved Web sources. For each data source, we consider up to four attributes for matching. The number of entities per source ranges from about 1,100 to more than 64,000; the size of the Cartesian product for the four tasks ranges from roughly 1.2 million (Abt-Buy task) to 168 million (DBLP-Scholar) entity pairs. We applied a simple blocking on a low string similarity threshold to reduce the search space to the numbers that Table 1 shows (up to 607,000 pairs). To determine the match quality, we further created the perfect match results with the cardinalities also shown in Table 1.

We chose the match tasks to represent a spectrum of different data characteristics and difficulty levels. We expect the first task to be of low difficulty because it deals with publication entities from two well-structured bibliographic data sources (the Digital Bibliography & Library Project [DBLP] and the ACM digital library) that are at least partially under manual curation. The selected DBLP and ACM entities cover the same computer science conferences and journals. The second match task requires matching DBLP publications with publications from the entity search engine Google Scholar (Scholar).

Table 1. Overview of evaluation match tasks.

Match task			Source size (number of entities)		Mapping size (number of correspondences)	
Domain	Attributes	Sources	Source 1	Source 2	Input mapping (blocking result)	Perfect result
Bibliographic	Title Authors	DBLP-ACM	2,616	2,294	494,000	2,224
	Venue Year	DBLP-Scholar	2,616	64,263	607,000	5,347
E-commerce	Name Description	Amazon-GoogleProducts	1,363	3,226	342,761	1,300
	Manufacturer Price	Abt-Buy	1,081	1,092	164,072	1,097

Scholar automatically extracts its publication entities from full-text documents crawled from the Web. This data has many quality problems — in particular, duplicate publications, heterogeneous representations of author lists or venue names, misspellings, and extraction errors. To obtain the Scholar data, we sent numerous queries on the publication title and venue names and stored the combined query results as our evaluation data set. We determined the perfect match result manually.

The e-commerce tasks deal with sets of related product entities from online retailers Abt.com, Buy.com (Abt-Buy task), and Amazon.com, as well as Google's product search service, accessible through the Google Base Data API (Amazon-GoogleProducts task). To obtain the perfect match result, we included only product entities with a valid Universal Product Code (UPC) in our data sets, which allows for unique product identification. Of course, the match strategies we planned to evaluate couldn't use these UPCs but only the attributes listed in Table 1 (product name, description, manufacturer, and price). In reality, many websites don't provide the UPC information, so entity matching can't rely on these in general.

We created the Abt, Buy, and Amazon data sets by selecting products from predefined categories. Based on the Amazon products, we generated the GoogleProducts data set by sending queries on the product name.

We use the common measures precision, recall, and F-measure to quantify the quality of entity match strategies with regard to the perfect match result.

Evaluation Results

We first discuss the results for our manually configured match strategy using a state-of-the art commercial match implementation, which serve as baseline results for our learning-based strategy evaluation. We then compare the random and ratio approaches' effectiveness for training selection and that of the four different learning methods.

Manual baseline strategies. To better assess the quality of the automatically generated, learning-based match strategies, we applied a state-of-the-art entity match system to our match tasks. (Due to license restrictions, we can't provide the evaluated system's name.) The approach has several parameters that we needed to configure. The most important is the *overall MinimumSimilarity* threshold. We consider an entity pair a match only if it has a similarity greater than or equal to this threshold. We can optionally specify additional *attribute-level similarity thresholds* for each attribute that is used for computing the entity similarity. Hence, the number of parameters grows with the number of attributes. Table 2 shows the precision, recall, and F-measure results for the four match tasks using either one or two attributes with standard configurations (0.5 for overall MinimumSimilarity, 0.0 for attribute MinimumSimilarity). For these tests, we used the first or first two attributes listed in Table 1 (publication titles and authors for the bibliographic tasks, product names, and descriptions for the e-commerce tasks). Table 2 reveals significant differences for the four match tasks. Whereas we could effectively solve

Table 2. Match accuracy for a state-of-the-art approach with default and tuned configuration.

	DBLP-ACM			DBLP-Scholar			Abt-Buy			Amazon-GoogleProducts		
Number of attributes	1	2	2	1	2	2	1	2	2	1	2	2
Precision (%)	94.9	96.9	97.6	74.1	77.5	77.5	78.4	90.6	66.3	78.6	82.4	61.7
Recall (%)	97.3	87.8	90.2	91.7	84.8	89.2	36.4	17.6	65.3	51.3	39.9	62.7
F-measure (%)	96.1	92.1	93.8	82.0	81.0	82.9	49.7	29.5	65.8	62.1	53.7	62.2

the first bibliographic match task (F-measure > 92 percent), the results for the other three tasks were much worse, especially for the e-commerce tasks. Furthermore, the default parameters resulted in a reduced match quality for two attributes compared to only one attribute for all four tasks, indicating a strong need for manually finding better parameter settings.

However, finding suitable parameter settings is very challenging, even for domain experts, due to the large number of possible parameter combinations. To find better baseline results than we do using the default parameters, we used Fever on smaller subsets of the match tasks (500 randomly selected entity pairs with a minimal string similarity, analogous to the random training selection approach) to find the best settings for the three similarity thresholds when using two attributes for matching. For each of the three MinimumSimilarity thresholds, we considered 11 values (0 to 1 in 0.1 steps), resulting in a total of 1,331 configurations that we evaluated for each of the four match tasks. For each task, we chose the configuration with the highest F-measure as the baseline strategy. The third column of Table 2 indicates the corresponding results for the entire data set ("tuned" for two attributes) for each match task. We observe that the tuned strategies always outperform the default configuration for two attributes. For the three more challenging tasks, the tuned strategies also outperform the default strategy on one attribute. The high tuning effort indicates that the reported results are rather optimistic for manually determined match strategies with state-of-the art implementations. The fact that the absolute match effectiveness remains comparatively low, especially for the e-commerce tasks, underlines that these are really challenging problems to deal with.

Random vs. ratio training selection. For the initial experiment on the effectiveness of automatically learned match strategies, we evaluated the two methods proposed for selecting training data: random and ratio. We considered the results for two training sizes of 50 and 500 selected entity pairs, representing a rather small-to-moderate labeling effort. We varied the minimal similarity threshold for the TFIDF similarity (on the first attribute listed in Table 1) from 0.3 to 0.8.

Figure 3 displays the F-measure results for the four match tasks DBLP-ACM, DBLP-Scholar, Abt-Buy, and Amazon- GoogleProducts and compares the random and ratio training selections. Figure 3a shows the results for labeling effort 50 and Figure 3b the results for labeling effort 500. We obtained the F-measure results with eight matchers and with SVM as the learner. For comparison, we also show the F-measure results for the manually determined baseline match configurations. The matchers used for learning operate on the same two attributes as the baseline strategy but apply one of the four similarity measures (Cosine, Jaccard, TFIDF, or Trigram) on them, resulting in eight matchers.

We first observe that even for the small training size of 50, the learned match strategies mostly outperform the baseline strategies, especially for the more difficult e-commerce tasks (about 14 percent improved F-measure values). Although the random and ratio approaches perform largely similarly, random is consistently somewhat less effective and more dependent on the chosen similarity threshold and training size. For higher similarity thresholds (≥ 0.6), random mainly selects matching entity pairs and thus provides few nonmatching pairs, making it difficult to learn how to identify nontrivial, nonmatches. Furthermore, the nonmatching entity pairs selected with a high threshold might be rare outliers, and we risk the learned model being overfitted to those special cases, preventing it from classifying other entity pairs correctly.

The ratio approach is generally better than random because it maintains a better balance between matching and nonmatching entity

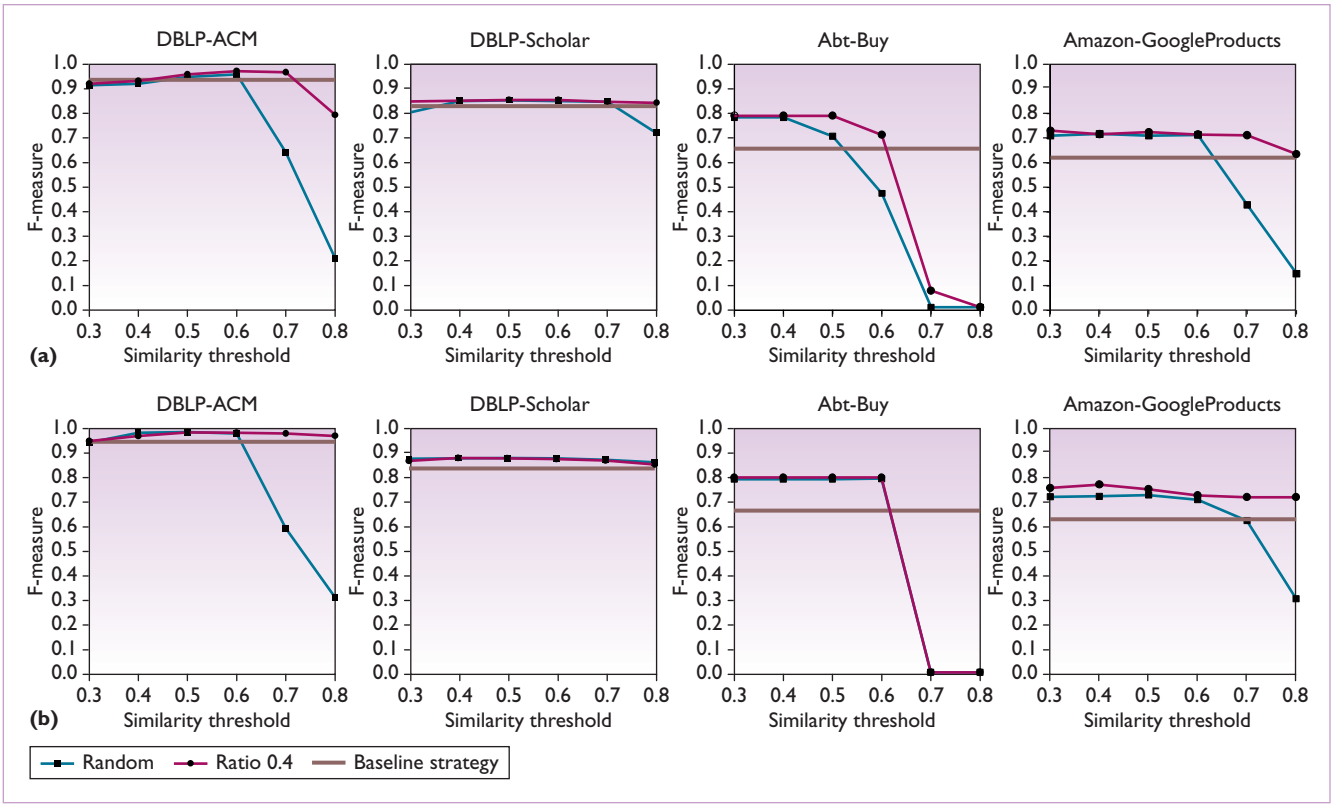

Figure 3. Comparison of random and ratio training selections using support vector machines (SVMs) with eight matchers. We looked at results for training sizes of (a) 50 and (b) 500 selected entity pairs.

pairs by eliminating entities from a randomly selected set of pairs. Although this reduces the remaining number of training data, our results show that this is more than offset by the better quality for learning. We experimented with different values for the ratio parameter and found rather stable results in the range from 0.2 to 0.5, with 0.4 as a good compromise value. Figure 3 shows that ratio is also relatively stable for similarity thresholds between 0.3 and 0.6, even for smaller training sizes. For Abt-Buy, a similarity threshold of ≥ 0.7 left almost no nonmatches due to a heterogeneous representation; ratio thus became unable to retain a sufficient number of training pairs.

Based on this experiment, we conclude that the ratio approach is effective for selecting training data for learning-based entity matching. In our further experiments, we'll use this strategy with a default setting of 0.4 for both the ratio and similarity threshold parameters.

Comparison of different learners. In this experiment, we wanted to evaluate the relative quality of the four learner strategies for determining a combined entity matching strategy: decision tree, logistic regression, SVM, and the multiple learning approach. We used the same eight matchers as in the previous experiment.

Figure 4 shows the F-measure results for the four match tasks achieved with the four learners and different labeling efforts (x-axis). The labeling effort varies between 20 and 500 entity pairs. We observe that all learners benefit from increasing the training size, especially for the Scholar-DBLP and Abt-Buy tasks. For all match tasks, the learner strategies could clearly outperform the baseline strategy in most cases, even for very small training sizes of 20 or 50 labeled entity pairs. For the maximum training size of 500, they could improve the F-measure results to about 97 percent (versus 94 percent for the baseline strategy) for DBLP-ACM, 91 percent (versus 83 percent) for DBLP-Scholar, 86 percent (versus 66 percent) for Abt-Buy, and 77 percent (versus 62 percent) for Amazon-GoogleProducts.

We observe that the three basic learners perform differently for the four match tasks so that no single basic learner consistently outperforms the others. For example, decision trees perform the worst for DBLP-ACM but the best for Abt-

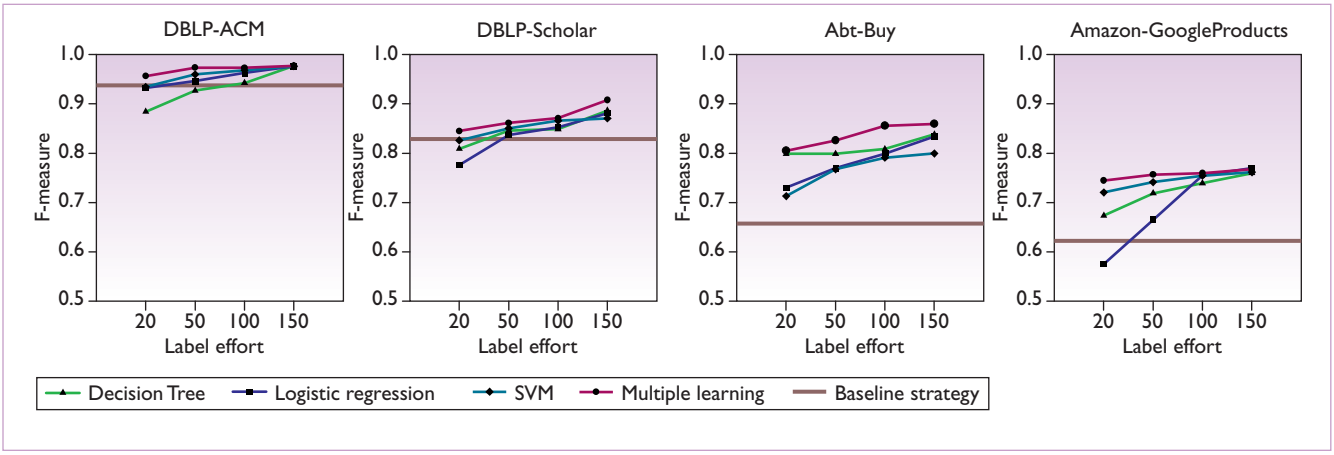

Figure 4. Comparison of different learners for matcher combinations on four match tasks under different labeling efforts (using ratio 0.4 training selection and eight matchers). The baseline results of a commercial entity matching approach are mostly outperformed even with small training sizes.

Buy. The decision tree and logistic regression approaches benefit most from more training data, whereas SVM performs relatively well even for small training sizes.

An important observation is that the rather simple multiple learning approach consistently performs best for all match tasks and training sizes. This shows that it can effectively combine individual basic learners' strengths and compensate for their weaknesses. Researchers have also demonstrated multiple learning approaches' effectiveness demonstrated in areas other than entity resolution.[13]

In additional experiments, we studied different matcher selections. The results indicated that more matchers don't necessarily improve match quality and tend to require more training.

Although the overall results of our study are encouraging, we see the need for further improvements in match quality, especially for e-commerce data. We therefore plan additional preprocessing steps to reduce data heterogeneity and the exploitation of matchers exploiting domain knowledge. Furthermore, we want to investigate the runtime efficiency and apply performance techniques such as the use of parallel entity matching.

References

1. C. Batini and M. Scannapieco, Data Quality: Concepts, Methodologies and Techniques, Data-Centric Systems and Applications Series, Springer, 2006.
2. A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios, "Duplicate Record Detection: A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 1, 2007, pp. 1–16.
3. H. Köpcke and E. Rahm, "Frameworks for Entity Matching: A Comparison," *Data & Knowledge Eng.*, vol. 96, no. 2, 2010, pp. 197–210.
4. N. Koudas, S. Sarawagi, and D. Srivastava, "Record Linkage: Similarity Measures and Algorithms," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2006, pp. 802–803.
5. M. Bilenko, S. Basu, and M. Sahami, "Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping," Proc. 5th IEEE Int'l Conf. Data Mining (ICDM 05), IEEE CS Press, 2005, pp. 58–65.
6. M. Bilenko and R.J. Mooney, "On Evaluation and Training-Set Construction for Duplicate Detection," KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation, 2003, pp. 7–12.
7. S. Chaudhuri et al., "Example-Driven Design of Efficient Record Matching Queries," *Proc. 33rd Int'l Conf. Very Large Databases* (VLDB 07), ACM Press, 2007, pp. 327–338.
8. H. Köpcke, A. Thor, and E. Rahm, "Comparative Evaluation of Entity Resolution Approaches with FEVER," *Proc. Very Large Databases Conf.*, ACM Press, 2009, pp. 1574–1577 (demo paper).
9. A. Thor and E. Rahm, "MOMA – A Mapping-Based Object Matching System," *Proc. 3rd Biennial Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 248–258.
10. H. Köpcke and E. Rahm, "Training Selection for Tuning Entity Matching," *Proc. Int'l Workshop Quality in Databases and Management of Uncertain Data* (QDB/MUD 08), 2008, pp. 3–12.
11. R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms, *Proc. 23rd Int'l Conf. Machine Learning* (ICML 06), ACM Press, 2006, pp.1 61–168.

12. I. Mierswa et al., "Rapid Prototyping for Complex Data Mining Tasks," *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, ACM Press, 2006, pp. 935–940.
13. H. Halteren, W. Daelemans, J. Zavrel, "Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems," *Computational Linguistics*, vol. 27, no. 2, 2001, pp 199–229.

Hanna Köpcke is the head of the object-matching workgroup of the WDI Lab and a PhD student at the University of Leipzig. Her research interests include data integration, entity resolution, and data mining. Köpcke has a Diploma in computer science from the University of Dortmund Germany. Contact her at koepcke@informatik.uni-leipzig.de.

Andreas Thor is a research scientist with the database group at the University of Leipzig and a visiting research scientist at the University of Maryland Institute for Advanced Computer Studies (UMIACS). His research areas deal with integration of Web data sources, specifically approaches for entity resolution, ontology alignment, and flexible integration architectures. Thor has a Diploma and a PhD in computer science from the University of Leipzig. Contact him at thor@informatik.uni-leipzig.de.

Erhard Rahm is a professor and the chair for databases at the University of Leipzig's Institute of Computer Science. His current research areas are data integration, metadata and ontology management, and bio databases. Rahm has a PhD in computer science from the University of Kaiserslautern. He supervises the WDI lab at the University of Leipzig, which is a third-party funded innovation lab on Web data integration. Contact him at rahm@informatik.uni-leipzig.de ; http://dbs.uni-leipzig.de.