Semi-supervised Instance Matching Using Boosted Classifiers

Mayank Kejriwal and Daniel P. Miranker

University of Texas at Austin
{kejriwal,miranker}@cs.utexas.edu

Abstract. Instance matching concerns identifying pairs of instances that refer to the same underlying entity. Current state-of-the-art instance matchers use machine learning methods. Supervised learning systems achieve good performance by training on significant amounts of manually labeled samples. To alleviate the labeling effort, this paper presents a *minimally supervised* instance matching approach that is able to deliver competitive performance using only 2% training data and little parameter tuning. As a first step, the classifier is trained in an ensemble setting using *boosting*. Iterative *semi-supervised learning* is used to improve the performance of the boosted classifier even further, by *re-training* it on the most confident samples labeled in the current iteration. Empirical evaluations on a suite of six publicly available benchmarks show that the proposed system outcompetes optimization-based minimally supervised approaches in 1-7 iterations. The system's average F-Measure is shown to be within 2.5% of that of recent supervised systems that require more training samples for effective performance.

Keywords: Instance Matching, Semi-supervised Learning, Boosting

1 Introduction

Instance matching is the problem of matching pairs of instances that refer to the same *underlying* entity [24]. It is an important preprocessing step in knowledge discovery and data mining algorithms [6], and is documented to have numerous applications in the Semantic Web community [24].

Current state-of-the-art instance matchers use a variety of machine learning techniques to achieve effective performance [5], [1], [27]. Many of these systems are *supervised*, and require sets of manually annotated samples to train their classifiers. This manual effort can be expensive, especially in open communities.

In recent years, *minimally supervised* approaches have been devised to alleviate extensive labeling effort [16],[15]. While such approaches perform reasonably in many cases, a comparative analysis shows that there is still a considerable gap between their performance and that of supervised systems [17]. An additional problem is that such systems require extensive parameter tuning, and the specification of a function called the *pseudo F-Measure* (PFM). Intuitively, the PFM serves as a proxy for the *true* F-Measure, with minimally supervised instance matchers heuristically attempting to optimize the PFM over unlabeled

(or sparsely labeled) data instead of the true (unknown) F-Measure [18]. A recent study found the PFM to be uncorrelated (and even *negatively* correlated) with the true F-Measure in several cases [17], raising concerns about whether currently defined PFMs are appropriate proxies.

This paper presents a *minimally supervised* instance matching system that offers a practical compromise between the two paradigms above. The proposed system expects a few input *seed* training samples to bootstrap itself. To maximize its performance on unseen data, the system employs a *meta-classification* strategy called *boosting* [7]. Boosting is a machine learning method that relies on weighting several *base* machine learning classifiers to build an *ensemble classifier*. Ensemble classifiers use weighted majority voting to classify samples, which is shown to improve performance on many challenging tasks [7].

The ensemble classifier in this paper is used for *probabilistic* instance matching, where the classifier scores each instance pair according to its likelihood of being a matching pair. Given the low degree of supervision, the overall output is not expected to have high quality. Instead, the system uses a small percentage of the most confidently labeled instance pairs to iteratively *self-train* itself in a *semi-supervised* fashion. The intent is to improve performance with each iteration, with large gains anticipated in the initial iterations.

To the best of our knowledge, this is the first minimally supervised instance matching system that combines boosting methods with iterative semi-supervised learning to achieve effective performance. The ensemble classifier is trained using the *AdaBoost* algorithm [21], and with a choice of two base classifiers, *random forests* and *multilayer perceptrons* [12], [23], both of which have been individually validated for instance matching [22], [27].

Evaluations on six benchmark datasets show that, using just 2% of the ground-truth (or 50 samples, whichever is less) for training, the proposed system with the multilayer perceptron as a base classifier outperforms, on average, state-of-the-art minimally supervised approaches, and performs competitively compared to fully supervised systems that use more training samples. Additionally, the best performance is consistently achieved within 1-7 semi-supervised iterations. The system is also shown not to require extensive parameter-tuning in order to achieve these benefits. Lastly, we show, through implementation, that the proposed system can be integrated seamlessly with state-of-the-art orthogonal components (e.g. *blocking*) that are required in a complete workflow.

2 Related Work

Instance matching is an extensively researched subject, and goes by many different names, including *record linkage*, *entity resolution*, *the merge-purge problem* and *data matching*, to name just a few [24], [3], [6]. A naïve instance matcher pairs every instance in the dataset with every other, and then scores the pair (as matching or non-matching) in an expensive *classification* phase [6]. The untenable quadratic complexity of this approach indicates a two-step workflow, with the first step designated as *blocking* [4]. Blocking places instances into (possibly

overlapping) clusters, either by *partitioning* the instance space in some manner [14], or by using an *inexpensive* clustering function called a *blocking key* [4], [8]. Instances *sharing* a cluster are paired and classified, leading to savings.

State-of-the-art systems that focus on the blocking aspect of instance matching include Limes and MultiBlock [14], [8], the latter being implemented in the *Silk* toolkit. Both these approaches depend on the link specification classifier being *known*. Section 3.1 describes why this supervised blocking approach is unsuitable for the present work. To the best of our knowledge, only two *classifier-agnostic* approaches have been proposed for schema-free RDF data. The first of these is the unsupervised *Attribute Clustering* (AC) algorithm, recently proposed by Papadakis et al. [20], and is used as the blocking module in this paper. Recently, a DNF (Disjunctive Normal Form) blocking scheme learner for RDF data was also proposed but is relatively more complex to implement [9], [10].

The classification step has also been extensively researched, with a survey of existing systems provided by Scharffe et al. [24]. Popular examples of supervised systems include FEBRL [5] and Marlin [1]; for a comparative evaluation, we refer the reader to the work by Köpcke et al. [11]. To the best of our knowledge, the only work that has considered a multilayer perceptron is a supervised evaluation effort by Soru and Ngomo [27]. Also, we are only aware of one (supervised) work that has implemented boosting in an instance matching architecture [22]. In contrast to either effort, this paper proposes a *minimally supervised* instance matcher that simultaneously incorporates boosting and iterative semi-supervised learning to improve performance. For a full treatment on boosting, the reader is referred to the seminal work by Freund and Schapire [7]. The book by Chapelle et al. comprehensively covers semi-supervised learning [2].

As earlier mentioned, current minimally supervised approaches optimize a pseudo F-Measure (PFM) function [18], or perform *active learning* [15], [16]. The proposed system is compared against these approaches in Section 4; they were also compared under different configuration settings in a recent evaluation effort [17]. Finally, two influential examples of *fully implemented* RDF-based instance matchers are RDF-AI and KnoFuss [25], [19].

3 Approach

The schematic of the full instance matching system is illustrated in Figure 1. Note that, while the dotted component (the classification step) in the figure constitutes the key innovation, it cannot be implemented in isolation. In addition to blocking, generating restriction sets (matching classes and properties between two files) is an important task in the schema-free RDF setting [15]. To maximize performance, we re-implement state-of-the-art pre-classification modules, with a preference for unsupervised, but empirically high-performing, approaches. Note that, in principle, a practitioner can always replace a module with their own. Experimentally, the modules below delivered good performance (Section 4).

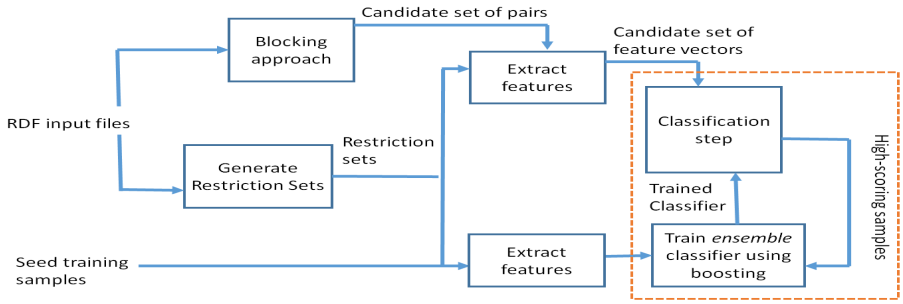

Fig. 1. The proposed instance matching system. The dotted component (the *classification step* of instance matching) is iteratively executed for a pre-defined number of self-training rounds, and constitutes the key innovation of the paper

3.1 Pre-Classification Steps

Blocking approach. Recall, from Section 2, that blocking is the first step of typical two-step instance matchers and can be thought of as a pre-processing clustering step that selects only a subset (of the Cartesian product of the full sets of instances) for further processing [4]. The goal is to avoid exhaustive pairwise comparisons of instances in the classification step. Let the set of instance pairs generated by the blocking step be denoted as the *candidate set*.

Blocking may generate the candidate set with or without the knowledge of the classifier (see Section 2). In a semi-supervised setting, the latter approach is advantageous. The main reason is that, since candidate set generation is independent of classification, it only needs to be executed *once*. Thus, only the classification step is subject to semi-supervised learning (the boundary of the dotted component in Figure 1), leading to computational savings.

Given this rationale, this paper uses the recently proposed *trigrams-based Attribute Clustering* (AC) algorithm for the blocking approach [20]. The approach is *unsupervised* and performs well empirically. The AC algorithm works by grouping properties (or *attributes*) after computing the overlap between the properties' value-sets using a trigram-based similarity score. Two instances share a cluster if they share tokens in any two properties that were grouped together. The candidate set is generated from the clusters using a suitable blocking algorithm, several of which were evaluated in the original paper [20]. Experimentally, the *block purging* algorithm[1] was found to work well for the proposed system.

Generating restriction sets. *Restriction sets* in the instance matching literature are typically defined as sets of *class* and *property alignments* between the input RDF files [15].

[1] Block purging eliminates clusters larger than a threshold value, with the premise that such clusters are the result of (non-discriminative) stop-word tokens [20].

Example 1: Suppose both input RDF files contain interlinked instances from two classes, *Addresses* and *People*. A restriction set would determine that these classes do not match, and would ensure that their instances are not paired with each other for further evaluations.

In practice, restriction sets tend to improve both quality *and* run-time, since they ensure that only *compatible* instance pairs are evaluated. Similarly, when extracting features from a given instance pair (described subsequently), the system is more effective if it has access to a set of matching property pairs. The *Raven* system was one of the first instance matchers to automatically deduce restriction sets from the data [15]. Experimentally, the solution was effective when there was considerable *extensional overlap* between two properties (for property alignment) or instance sets (for class alignment), assumptions that were found to hold for the benchmarks in this paper. For more heterogeneous datasets, a sophisticated type-inference algorithm (e.g. *Typifier* [13]) may be a safer option.

Extracting features. An instance matching classifier does not directly take an instance pair as input. Instead, the pair is first converted into a real-valued feature vector. Let a *feature extraction function* be defined as a function that accepts a pair of strings as input and outputs a real-valued number. Given a set G of such functions and a set Q of property alignments output by the restriction set generator, $|Q||G|$ features can be extracted for each compatible instance pair, by applying each function in G to the *property values* corresponding to an alignment in Q.

Example 2: Consider two independent datasets describing people and an alignment *(Home-Address, Residence)* between two of their respective properties. Given an instance pair (e_1, e_2) from the datasets, the simple feature extraction function *CommonToken* would compare their addresses and return 1 if they share a common token, and 0 otherwise.

The choice of G is an important determinant of overall instance matching performance [3]. Existing instance matchers typically include token-based and string-based functions as features [5], [1]. Numeric features have also been found to improve performance, especially if dates and other numeric data regularly occur in the files [22]. A comprehensive text by Christen evaluated *phonetic* features, and found them to be quite effective [3]. Drawing on these efforts, we implement 28 feature extractors for the proposed system, including 2 numeric features, 8 string and token-based features, and 18 phonetic features. Many of these features are efficiently implemented in the FEBRL package [5]; more details and examples are provided on our project website [2].

3.2 Classification Step

Algorithm 1 contains the pseudocode for the classification step. In addition to the base classifier M, the algorithm takes as arguments seed training sets

[2] `https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/semi-supervised-im-using-boosting`

Algorithm 1 Classification step

Input: Seed training sets (comprising *feature vectors*) of positive and negative samples D and N resp., Candidate Set Γ, Base Classifier M, Iteration rounds num, Positive factors for positive and negative samples $factor_D$ and $factor_N$ resp.
Output: Ranked list L of pairs in Γ

1. Initialize list L of size $|\Gamma|$
2. Initialize $num_D := |D|$
3. Initialize $num_N := |N|$
4. Train classifier M with the *AdaBoost* ensemble method using D and N as training sets; let the trained classifier model be denoted as M'
5. Initialize $count := 0$
6. **while** $count < num$ **do**
   Score each pair in Γ using M' and place pair in L
   Sort L in ascending order using the scores as sorting keys
   $num_D := num_D \times factor_D$
   $num_N := num_N \times factor_N$
   Repeat step 4 by using the first num_D elements in L as positive training examples, and last num_N elements in L as negative training examples
   $count := count + 1$
7. **end while**
8. **return** L

of matching and non-matching instance pairs and the candidate set Γ, with each instance pair in these sets converted to a feature vector. Finally, three parameters (num, $factor_D$ and $factor_N$) are used to control semi-supervision and are subsequently described.

Freund and Schapire first described boosting as 'the general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb' [7]. Thus, boosting is an ensemble-based method that seeks to train and combine several instances of a *base* classifier to obtain a final strong *ensemble* classifier. A popular implementation of a boosting algorithm, and the one used in this paper, is *AdaBoost* [21]. AdaBoost works by dynamically placing higher weights on training samples that are misclassified by the current classifier in each boosting round. The *committee* of classifiers thus trained are in turn weighted according to their overall performance on the training set, with the weighted committee constituting the *ensemble classifier*. During testing, the ensemble classifier scores a feature vector from the candidate set by computing an appropriately normalized weighted sum of scores.

To illustrate this process, suppose that a trained instance M' of the base classifier M is configured to output a confidence score $score(M')$ for a particular classification. Assuming that the ensemble classifier is a weighted committee of m trained models, $w_1 M'_1 + \ldots + w_m M'_m$, the confidence score of the ensemble classifier on a feature vector is $w_1 score(M'_1) + \ldots + w_m score(M'_m)$.

Once the ensemble classifier is trained, every feature vector in Γ is scored in this manner and the sorted list L is compiled. Note that, if the training sets

are large enough, the parameters of the base classifier and also AdaBoost can be determined through grid-search and cross-validation. In the present task, the training sets are assumed to be small, typically of the order of 2% of the ground-truth or 50 samples, whichever is less. In many benchmarks, 2% of the ground-truth constituted fewer than even 10 training samples. In such situations, parameter tuning is viable. Instead, the goal is to achieve good generalization for reasonable *default* values of the parameters.

To accomplish this and avoid overfitting, Algorithm 1 employs *semi-supervised learning* to iteratively *self-train* the classifier on (previously) unlabeled samples for *num* iterations. It is also possible to devise alternate convergence choices, but this issue is left for future work. Concerning how many *more* samples the system should self-train on in each new iteration, Algorithm 1 uses the parameters $factor_D$ and $factor_N$ for this purpose. It is typical to set $factor_D = factor_N (= factor)$. In this paper, we adopt an *aggressive* strategy and set $factor = 2$. Intuitively, such a strategy leads to stable performance in only a few iterations, but risks introducing more noise into the system. $factor$ can also be used to set num, assuming that $x\%$ of the ground-truth was used for bootstrapping the system. Consider that, in the first self-training round, $factor * x\%$ (positive and negative) samples are used, followed by $factor^2 x\%$ in the next round (and so on). It is reasonable to assume that $factor^{num-1} x$ should not be allowed to exceed 100%. Setting $x = factor = 2$ indicates[3] that $num \leq 7$.

In early experiments, we found $factor = 2$ (but with the caveat in the footnote) and $num = 7$ to yield a good compromise between noise and convergence, and assume these values in the rest of this work. A detailed analysis of alternate parameter settings is a topic for future work.

In summary, while boosting starts from a *weak classifier* and attempts to strengthen it by dynamically *re-weighting* the training set, semi-supervised learning starts from a *small training set* and attempts to *grow* it iteratively by exploiting confidence scores.

Note that the performance of boosting depends on the base classifier M. In this paper, we consider both *random forests* and *multilayer perceptrons* as base classifiers. A random forest is a committee of bootstrap-aggregated (or 'bagged') decision trees, and is known to make decision tree performance more robust on noisy data [12]. A multilayer perceptron (MLP) is a *feedforward artificial neural network* that can distinguish data that is not linearly separable, unlike the original perceptron model [23]. As Section 4 will illustrate, the two classifiers offer different tradeoffs. MLPs tend to perform better empirically on challenging tests, but take more time to train, even on small training sets. In contrast, random forests have fast training times, but may suffer from low performance on difficult test cases [27].

The performance of semi-supervised learning depends both on the performance of the boosted classifiers and the number of incorrectly labeled samples

[3] Note that $2^7 = 128\%$. To prevent this extra source (28%) of noise, the seventh iteration of Algorithm 1 sets $factor$ to 100/64=1.5625. More generally, Algorithm 1 can be implemented to take x as a parameter, and to enforce $factor^{num-1} x \leq 100\%$.

in each of the self-training sets. If the system does not perform well initially, semi-supervision is likely to degrade performance further. The next section will show that, in many cases, semi-supervised learning and boosting can be used to offset each other's disadvantages. Semi-supervised learning helps to compensate for the overfitting problems often caused by boosting on small training sets [12], while boosting helps to compensate for the incorrect labels ('noise') introduced by semi-supervised learning.

4 Experiments

4.1 Data

Six benchmarks are used to evaluate the system. The first three, *Persons 1*, *Persons 2* and *Restaurants* were publicly released by the 2010 Instance Matching Evaluation Initiative (or IAEI), conducted as part of the annual Ontology Alignment Evaluation Initiative[4] (OAEI). These cases are 'easy' in that *supervised* systems have been shown to achieve over 95% F-Measures on all of them [11]. This is not true of *minimally supervised* approaches, as Section 4.5 will demonstrate. For this reason, the three benchmarks provide an interesting test for the proposed (minimally supervised) system.

The other three real-world benchmarks are designated as *ACM-DBLP, Amazon-GoogleProducts* and *Abt-Buy*[5]. *ACM-DBLP* covers the bibliographic domain and is relatively clean, with 2617 and 2295 instances in the source and target respectively, and a ground-truth set of 2224 matching pairs. The other two datasets cover e-commerce instances and are known to be difficult even in supervised settings [11], [27]. *Amazon-GoogleProducts* links 1363 source instances to 3226 target instances via 1300 matches, while *Abt-Buy* links 1081 source instances to 1092 target instances via 1097 matches.

The six described benchmarks were specifically chosen because, along with covering several domains, they enable comparing the proposed system to the *best* reported results of at least four other state-of-the-art approaches that were recently evaluated on them [27], [11], [17].

4.2 Implementation

Random forests, multilayer perceptrons and the AdaBoost algorithm are already implemented in the Java Weka API[6], which were used for these experiments. Given the small size of the seed training set, Weka's default parameter values are used without any special tuning. Section 4.6 discusses this issue further. Finally, all programs were implemented serially in Java on a 32-bit Ubuntu machine with 3385 MB of RAM and a 2.40 GHz Intel 4700MQ i7 processor.

[4] `http://oaei.ontologymatching.org/2010/im/index.html`. We did not use the 2014 IAEI benchmarks because, at the time of writing, their ground-truths were unavailable, and they were not evaluated by competing instance matching baselines.

[5] Available at `http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution`

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

4.3 Pre-classification results

The re-implemented Raven restriction set generator was found to yield perfect class and property alignments for all the benchmarks. Similarly, the trigram-based Attribute Clustering blocking approach yielded perfect candidate set recall for the three IAEI benchmarks. On the real-world datasets, the approach was not able to achieve perfect recall, but still performed reasonably. Specifically, the *Abt-Buy* and *ACM-DBLP* candidate sets covered 95.44% and 97.43% of the ground-truth respectively. Recall on *Amazon-GoogleProducts* was the lowest (83.54%) owing to the difficulty of the dataset. Note that this bounds the maximum recall that can be achieved by the classification step, which penalizes its maximum possible F-Measure. The next section discusses this issue further.

In terms of run-time, all pre-classification steps were found to execute in a total of less than 3.5 seconds for all the benchmarks. Similar to existing instance matchers, this time was negligible compared to classification time (Section 4.6).

4.4 Classification metrics

The metrics, *precision*, *recall* and their $F_1 - Measure$, were used for evaluating performance. Denoting the set of returned results as R and the ground-truth as G, the precision is defined by the formula $|R \cap G|/|R|$, while the recall is defined by the formula $|R \cap G|/|G|$. The $F_1 - Measure$ (henceforth denoted simply as the *F-Measure*[7]) is given by *2*Precision*Recall/(Precision+Recall)* and quantifies precision-recall tradeoff.

The previous section noted that, on *Amazon-GoogleProducts* (and to a lesser extent, *Abt-Buy* and *ACM-DBLP*), candidate set recall was imperfect, which implies that the maximum achievable classification F-Measure is strictly below 100%[8]. For these systems, the reported F-Measures are only *pessimistic* estimates, since it is improbable that the classifier would have labeled *all* of the missing (matching) pairs incorrectly. To realistically estimate the true F-Measure, we *re-weight* the pessimistic F-Measure F_p using the formula $100 * FM_p/FM_{max}$, where FM_{max} is the maximum possible F-Measure achievable under the current candidate set (e.g. 91.03% on Amazon-GoogleProducts). Where applicable, both F-Measures (pessimistic and re-weighted) are reported.

4.5 Classification Results

For both base classifiers (random forests and multilayer perceptrons), the precision-recall tradeoff offered by the proposed system is evaluated against two baselines. The first baseline is the base classifier itself, trained on the same samples as the proposed system, but without boosting or semi-supervised learning. The second

[7] The general F-Measure formula is parametrized by a quantity, β. In the case of the $F_1 - Measure$, $\beta = 1$.

[8] For example, the maximum achievable classification F-Measure on *Amazon-GoogleProducts* is $2 * 83.54 * 100/(83.54 + 100) = 91.03\%$, since maximum achievable recall is the candidate set recall, 83.54%.

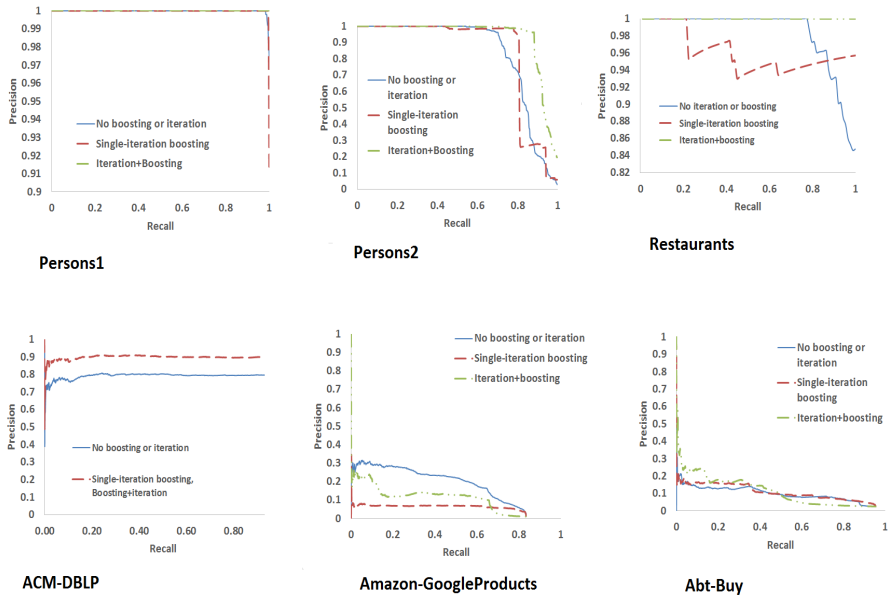

Fig. 2. The best precision-recall results of the proposed system (over seven iterations) against the two baselines, when using a random forest as the base classifier. On *Persons 1*, the three curves are *near*-coincidental, while in the case of *ACM-DBLP*, the best performance of the proposed system was achieved in the first iteration itself (hence, two curves *are* coincidental). Note the change in Y-axis scale for *Persons 1* and *Restaurants*

baseline is similar to the first, except that boosting (but not semi-supervised learning) is used. For each benchmark, the precision of both baselines is plotted against the recall. Given that the proposed system is evaluated over seven iterations, we plot (for each benchmark) the precision-recall curve for the iteration in which the proposed system achieved the highest F-Measure.

Figure 2 shows the results for the random forest base classifier. On *Persons 1*, all three systems performed equally well, achieving nearly 100% F-Measure. On both *Persons 2* and *Restaurant*, the proposed system either equals or outperforms the other two systems for all recall values. Interestingly, in the case of *Restaurants*, we note that although the second baseline (using just the boosted classifier) outperforms the first baseline in terms of the highest F-Measure achieved, the latter offers better tradeoff at *lower* recall values.

On the other three benchmarks, the differences between the systems are not as apparent. On *ACM-DBLP*, boosting affected performance positively, but semi-supervised learning did not, since the best (proposed) system performance was achieved in the first iteration itself. On *Amazon-GoogleProducts*, boosting clearly degrades performance, but the semi-supervised learning is able to partially compensate for it. On *Abt-Buy*, the three systems are again nearly identical, but unlike on *Persons 1*, none of the systems perform well. As earlier noted in

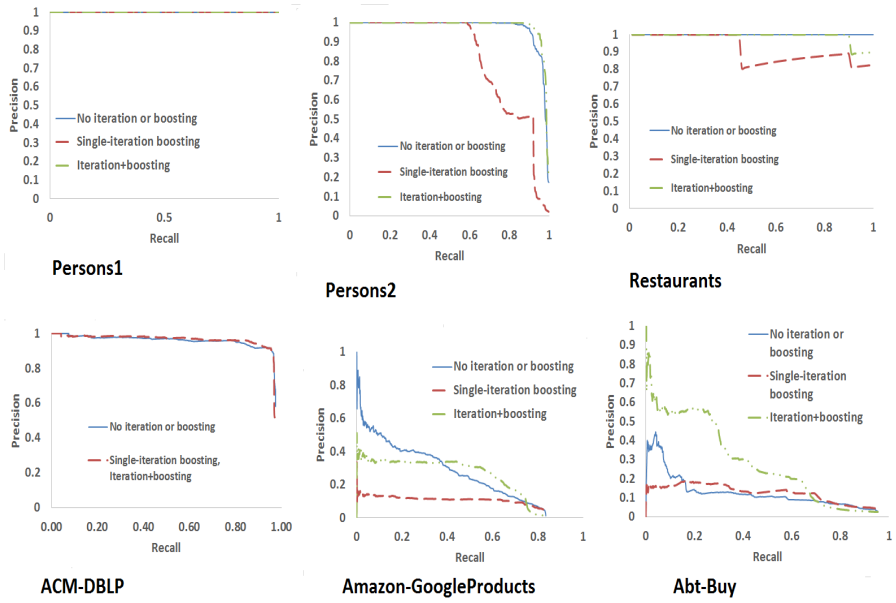

Fig. 3. The best precision-recall results of the proposed system (over seven iterations) against the two baselines, when using a multilayer perceptron as the base classifier. On *Persons 1*, the three curves are near-identical. Similar to Figure 2, the best result of the proposed system, in the case of *ACM-DBLP*, was achieved in the first iteration

Section 4.1, both Abt-Buy and *Amazon-GoogleProducts* are challenging cases, since even supervised systems performed relatively poorly on them [11], [27].

Figure 3 shows the results for the case where the base classifier is a multilayer perceptron (MLP). For the first four benchmarks, the findings are similar to those in Figure 2, with an exception in the case of *Restaurants*, where the proposed system shows a slight dip at the end and the base classifier ends up with the best F-Measure (100%). Otherwise, the highest F-measure is always achieved by the proposed system, an observation that is most apparent in the case of *Abt-Buy*. Another interesting finding is that the non semi-supervised boosted classifier consistently performs *worse* than the base classifier. We believe that this attests to the expressive strength of MLPs over random forests. Boosting the MLP makes it more prone to overfitting, but the semi-supervised learning compensates for it. As in the case of *Amazon-GoogleProducts* in Figure 2, this illustrates the utility of combining both techniques with an expressive base classifier.

Comparison to minimally supervised approaches Table 1 lists the highest F-Measure scores achieved by the proposed system over the seven iterations that were conducted and compares it to three state-of-the-art minimally supervised approaches that were also evaluated on the six benchmarks. As noted earlier in Section 2, these approaches also seek to minimize supervision by optimizing

Table 1. A comparison of the highest F-Measures achieved by the proposed system (multilayer perceptron-based) to those of other minimally supervised approaches

Test Case	**Linear**	**Boolean**	**Genetic**	**Proposed (pessimistic)**	**Proposed (re-weighted)**
Persons1	**100.00%**	99.50%	**100.00%**	**100.00%**	**100.00%**
Persons2	41.45%	59.12%	37.04%	**97.19%**	**97.19%**
Restaurants	88.56%	88.56%	88.56%	**94.68%**	**94.68%**
ACM-DBLP	**97.96%**	97.46%	97.71%	93.42%	94.65%
Amazon-GP	**49.08%**	39.97%	43.11%	39.13%	42.98%
Abt-Buy	**48.60%**	37.66%	45.08%	36.27%	37.14%
Average	70.94%	72.18%	68.58%	**76.78%**	**77.77%**

a function called a pseudo F-Measure (PFM) in a variety of hypothesis spaces including *linear*, *boolean* and *genetic* (Table 1). The *Raven* system attempts to optimize a user-provided PFM using linear and boolean classifiers and tuned parameter values [15], while *Eagle* exploits genetic algorithms [16]. For fairness, only the best results achieved by these systems are listed in Table 1. For the proposed system, the best results achieved with the multilayer perceptron as base classifier are shown, owing to its relatively superior performance[9]. The table show that, on average, even the pessimistic estimate exceeds the next best (the Raven boolean classifier) system performance by over 4.5%. If the random forest-based classifier is used on *Restaurants*, the difference widens by about 1% (see previous footnote).

Comparison to supervised approaches Soru and Ngomo recently evaluated several base classifiers in a supervised setting using 10-fold cross validation [27]. We mentioned in Section 2 that that was the only work, to the best of our knowledge, that attempts using multilayer perceptrons for the instance matching task. Without ensemble methods or semi-supervised learning, the average highest F-Measure that was achieved on the six benchmark test cases was 79.25%. If the re-weighted approach is assumed, the proposed system (multilayer perceptron-based) is within 1.5% of this average, from the data in Table 1, and even the pessimistic approach is within 2.5%. A similar finding applies when analyzing[10] the evaluations conducted by Köpcke et al. [11] on two other supervised SVM-based instance matching systems, FEBRL and Marlin [5], [1].

FEBRL and Marlin were not tested on the IAEI benchmarks since they were originally designed to solve the *record linkage* problem for Relational Databases (RDBs) [5], [1]. On the *ACM-DBLP* benchmark, the best configurations of both systems performed well, achieving near-perfect F-Measure scores. On *Amazon-GoogleProducts*, FEBRL achieved between 30-40% F-Measure even when trained on 100 training samples, while the F-Measure achieved by the best configuration of Marlin (when trained on 100 samples) approached 50%. On *Abt-Buy*, FEBRL

[9] The exception was *Restaurants* where the random forest achieved 100% best FM.
[10] The reference for this claim is Figure 3 (on page 6) of the original paper [11].

achieved about 20% F-Measure after being trained on 100 samples, while Marlin achieved above 60% F-Measure on its best configurations.

To conclude, the proposed system slightly outperforms FEBRL on the real-world benchmarks (despite being trained on only half the training set), but is outperformed by Marlin. The literature also indicates that these systems required careful selection of parameters and models, and do not include components for restriction set generation and unsupervised blocking. It is unclear if these systems can be adapted as instance matchers for schema-free RDF data.

4.6 Discussion

We conclude this section with a discussion of the run-times, as well as the dependence of the system on the parameter *num* (the number of iterations) in Algorithm 1. Although the multilayer perceptron (MLP) outperformed (by a large margin) the random forest base classifier for the more challenging test cases (*Amazon-GoogleProducts* and *Abt-Buy*), the difference was less severe on the other test cases. In the case of *Persons 2* and *Restaurants*, both methods performed equally well. The reason why this observation is important is because the MLP had much higher run-times than the random forest. On *Restaurants*, for example, the random forest-based system had run-times ranging from 2-5 seconds (for the entire classification step) depending on the iteration. The MLP-based system achieved run-times ranging from 17 seconds (for the first iteration) to almost 20 minutes (for the final iteration). Similar observations were noted for the other datasets[11]. This confirms earlier findings that the MLP can be slower by 1-2 orders of magnitude, and has a direct dependence on the size of the training set [27]. Given this disparity in run-times between the two classifiers, the random forest is clearly a better base classifier choice for the IAEI benchmarks, and considering only the slight performance penalty, *ACM-DBLP* as well.

Note that it was not always the case that the best performance was achieved in the last iteration. For *ACM-DBLP* in particular, Figures 2 and 3 show that the best performance was achieved in the first iteration. On average, we found that the performance tended to peak on or before the *fifth* iteration[12], after which it slowly started declining. We hypothesize that this is due to the boosted classifier getting overfitted on the training data after a certain number of iterations. Future work will test this hypothesis through more evaluations.

Finally, note that the proposed classifier was only tested with default parameter settings, because the seed training sets were too small to perform grid-search or cross validation. Thus, it is reasonable to assume that the performance of the system can be improved even further if a good parameter-tuning methodology can be devised, without using an extensive validation set. We believe that this is an important issue for future work.

[11] Supplemental experimental results are noted on the project website (footnote 2).
[12] This corresponds to $2^5 = 32\%$ of the ground-truth (assuming no re-training noise).

5 Conclusion and Future Work

This paper presented a minimally supervised instance matching system that combines semi-supervised learning with the AdaBoost algorithm to achieve effective performance. Using a multilayer perceptron as the base classifier with default parameter configurations, the system outperformed, on average, competing minimally supervised approaches by over 4.5%. It also came within 2.5% F-Measure (using a pessimistic estimate) of the performance of fully supervised approaches that use larger training sets and 10-fold cross-validation to achieve the same level of performance. Drawing on prior evaluations from the literature, the system was also found to be competitive with the state-of-the-art FEBRL instance matcher. Along with these results, the low run-times make the system a promising candidate for off-the-shelf schema-free RDF instance matching.

Future work will seek to efficiently apply the findings to large-scale datasets, which raises some new challenges, including the efficiency of semi-supervised methods and the large training times of multilayer perceptrons. Addressing these challenges is clearly important, especially for large-scale efforts such as Linked Open Data[13] [26].

References

1. M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.
2. O. Chapelle, B. Schölkopf, A. Zien, et al. *Semi-supervised learning*, volume 2. MIT press Cambridge, 2006.
3. P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.
4. P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.
5. P. Christen, T. Churches, and M. Hegland. Febrl–a parallel open source data linkage system. In *Advances in knowledge discovery and data mining*, pages 638–647. Springer, 2004.
6. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.
7. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.
8. R. Isele, A. Jentzsch, and C. Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *WebDB*, 2011.
9. M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 340–349. IEEE, 2013.

[13] `linkeddata.org`

10. M. Kejriwal and D. P. Miranker. A two-step blocking scheme learner for scalable link discovery. *Ontology Matching*, page 49, 2014.
11. H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.
12. A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
13. Y. Ma, T. Tran, and V. Bicer. Typifier: Inferring the type semantics of structured data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 206–217. IEEE, 2013.
14. A.-C. N. Ngomo. A time-efficient hybrid approach to link discovery. *Ontology Matching*, page 1, 2011.
15. A.-C. N. Ngomo, J. Lehmann, S. Auer, and K. Höffner. Raven–active learning of link specifications. In *Proceedings of the Sixth International Workshop on Ontology Matching*, pages 25–37. Citeseer, 2011.
16. A.-C. N. Ngomo and K. Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications*, pages 149–163. Springer, 2012.
17. A.-C. N. Ngomo and K. Lyko. Unsupervised learning of link specifications: deterministic vs. non-deterministic. In *OM*, pages 25–36, 2013.
18. A. Nikolov, M. dAquin, and E. Motta. Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications*, pages 119–133. Springer, 2012.
19. A. Nikolov, V. Uren, E. Motta, and A. De Roeck. Handling instance coreferencing in the knofuss architecture. 2008.
20. G. Papadakis, E. Ioannou, T. Palpanas, W. Nejdl, et al. A blocking framework for entity resolution in highly heterogeneous information spaces. 2012.
21. G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Machine learning*, 42(3):287–320, 2001.
22. S. Rong, X. Niu, E. W. Xiang, H. Wang, Q. Yang, and Y. Yu. A machine learning approach for instance matching based on similarity metrics. In *The Semantic Web–ISWC 2012*, pages 460–475. Springer, 2012.
23. D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *Neural Networks, IEEE Transactions on*, 1(4):296–298, 1990.
24. F. Scharffe, A. Ferrara, A. Nikolov, et al. Data linking for the semantic web. *International Journal on Semantic Web and Information Systems*, 7(3):46–76, 2011.
25. F. Scharffe, Y. Liu, and C. Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.
26. M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014*, pages 245–260. Springer, 2014.
27. T. Soru and A.-C. N. Ngomo. A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 41–44. ACM, 2014.