Abstract

This article presents an unsupervised system that performs instance matching between entities in schema-free Resource Description Framework (RDF) files. Rather than relying on domain expertise or manually labeled samples, the system automatically generates its own heuristic training set. The training sets are first used by the system to align the properties in the input graphs. The property alignment and training sets are used together to simultaneously learn two functions, one for the blocking step of instance matching and the other for the classification step. Finally, the learned functions are used to perform instance matching. The full system is implemented as a sequence of components that can be iteratively executed to boost performance. Evaluations on a suite of ten test cases show individual components to be competitive with state-of-the-art baselines. The system as a whole is shown to compete effectively with adaptive supervised approaches.

Keywords: Instance Matching, Unsupervised System, Schema-free data, Linked Data, Automatic Training Set Generation, Feature Selection, Property Alignment, Modularity

1. Introduction

Instance matching is the problem of identifying pairs of entities (equivalently, instances) that refer to the same underlying entity [1]. In the Semantic Web, the results of instance matching are used to specify owl:sameAs links between pairs of instances in one of more input datasets represented using the graph-based Resource Description Framework (RDF) data model [2]. An example is illustrated in Figure 1. The problem has important applications over structured [3], semistructured [4], and unstructured data models [4], and goes by many different names, including record linkage [5], deduplication [6] the merge-purge problem [7], data linking [8], entity resolution [9], link discovery [10] and co-reference resolution [11].

A typical instance matching workflow runs in two steps and is supervised [6]. The first step, called blocking, clusters entities into sets of overlapping blocks using a function called a blocking scheme. Entities sharing a blocking are paired and become candidates for further evaluations by an expensive link specification function in the classification step [11]. When blocking, the vast majority of non-matching pairs are discarded, leading to savings over the quadratic one-step approach where all possible entity pairs are naively evaluated by the link specification function.

In current state-of-the-art instance matchers (see Section 2), either the blocking scheme or the link specification function needs to be provided by a domain expert, or learned from manually labeled training examples. Either approach optimistically assumes a human in the loop.

To address the issue of supervision, this article presents a three-step approach that can perform instance matching in an unsupervised fashion. Figure 2 illustrates the principal components of the system. In addition to the two typical steps of blocking and classification, an unsupervised learning step (Step 0 in Figure 2) is included in the pipeline. The goal of the step is to generate the blocking scheme and link specification function required by the conventional two-step instance matcher without subsuming the two-step matcher in terms of time and space complexity.

Step 0 relies on a component called a training set generator (TSG) to automatically generate its own positive and negative training samples using near-linear time heuristics (see Section 4.2). Because the procedure is heuristic, the generated training set is expected to contain a small number of incorrectly labeled samples. The rest of the system must be designed to accommodate this noise.

The generated training set is input to a component called the property aligner (Section 4.3). Consider Figure 1 again for an example of what constitutes a property alignment. There is an evident partial match between the property labels d1:hasWife and d2:spouse of the two graphs. While the matching relation in this example has subsumption semantics, the property aligner is designed using hybrid techniques in order to maximize recall and detect partial alignments that may not necessarily have well-defined semantics. The alignment set is used by a feature generator to convert each entity pair in the heuristic training set into a feature vector, with the size of a vector being directly proportional to the cardinality of the alignment set. Intuitively, a high-quality, compact alignment set is desirable in order to control the combinatorial explosion of the feature space (Section 4.4). The feature vectors output by the feature generator are simultaneously used to learn the two required instance match-
Figure 1: An example of instance matching over two RDF graphs. The goal of an instance matcher is to match the three instances in Graph $G_2$ to their counterparts in Graph $G_1$, and declare owl:sameAs triples connecting the three instance pairs.

Figure 2: An unsupervised three-step instance matcher with initial inputs and final outputs in red/italics. This article describes and evaluates the Unsupervised Learning step, namely, the blocking scheme and the link specification function. These learning procedures are detailed in Section 4.5. Finally, the two learned functions output by the unsupervised learning step are used in a standard two-step instance matching workflow that includes only the blocking and classification steps (Section 4.6).

In practice, the inclusion of Step 0 in the pipeline is shown to allow the full instance matching system to bootstrap itself at low cost. In an iterative procedure, the most confident links output by the system after a first run can be fed back into the feature generator to output new feature vectors and re-boot the learning (and subsequent instance matching). Even one iteration is shown to yield empirical improvements over a single-pass execution.

The full system does not assume the input data files to have associated schemas or ontologies, in contrast to approaches that rely on ontology and schema matching before linking instances (see Section 2). Schema-free data is known to be common on the Web of Linked Data, which has continued to grow since starting from merely twelve datasets in 2007 [12]. The automatic linking of entities on Linked Open Data (LOD) has emerged as an important problem in recent years [10]. The system is designed to deal with the heterogeneities typically associated with linked data, including property heterogeneity and the presence of multiple instance types in a single RDF file [13]. Experiments on a recently released benchmark show that the system is gracefully able to accommodate multilingualism.

The full system in Figure 2 is empirically evaluated (see Section 5) on a suite of ten instance matching test cases, and at least one unsupervised configuration of the system is shown to outperform or be at par with supervised SVMs trained on manually labeled samples on six of the ten cases. Individual modules in the unsupervised learning step are shown to compete effectively with current popular alternatives. The experiments also show that the run-time of the unsupervised learning step is subsumed by that of the classification step, and can be executed
with minimum overhead.

2. Related Work

Instance matching has emerged as an important research problem in the structured [6], semistructured [8] and unstructured data communities [4]. It is common to separate instance matching efforts in the unstructured community (where it is commonly denoted as co-reference resolution [4]) from those in the other communities, owing to the unique natural language processing needs of the former. Thus, the unstructured case is not considered further in this discussion.

The two-step formulation is the favored approach in the literature for performing large-scale instance matching [14]. Comprehensive surveys covering techniques in instance matching have been provided by multiple authors, including Elmagarmid et al. [6], Winkler [5], Köpcke and Rahm [15], Ferraram et al. [8] and Wolger et al. [1]. Due to the longevity of instance matching and its many applications [3], both blocking and classification have witnessed individual research attention. In recent years, as the state-of-the-art has steadily become more sophisticated, it has become the norm to only cover one part of the two-step instance matching pipeline in an individual work [6], [11]. The book by Christen is a good reference for the overall instance matching task [14].

Instance matching continues to be an evolving research area in the Semantic Web, with proposed applications including knowledge graph identification [16], semantic search [17] and populating entity name systems [18]. Many different systems have been proposed over the years [8]. Some of these systems are domain-specific, such as the RKB’s Consistent Reference Service (RKB-CRS), which determines equivalent URIs within the academic domain [19], and GNAT, which is designed specifically for the music domain [20]. Among domain-agnostic (also denoted as universal) systems, supervised and unsupervised approaches for both blocking and classification exist, as described below.

2.1. Supervised Systems

Among supervised systems, LIMES [21], Silk [10], HELIOS [22] and HYPO [23] implement the two-step instance matching approach by expecting a link specification function to be provided to the system. The tools expect the specification to obey certain properties and use those properties to perform blocking. For example, LIMES and HYPO rely on a metric specification and use the triangle inequality to efficiently partition entities into blocks. Silk accepts non-metric specifications but requires additional functional inputs [24].

Several efforts have focused on learning link specification functions by using supervised machine learning techniques, including genetic programming [25], active learning [26] and adaptive classifiers such as multilayer perceptrons and Support Vector Machines [27, 28, 29]. Another example of a supervised system is ObjectCoref, which uses ‘seed’ owl:sameAs links already present in the dataset to bootstrap itself and discover more links [30]. Apart from some initial supervision, ObjectCoref also depends on semantics to perform matching, and is not schema-free. More recently, we developed a minimally supervised schema-free instance matcher that combines the classic machine learning techniques of boosting and semisupervised learning to achieve high performance using just 2% of the ground-truth for training [31]. One of the drawbacks of the system was high training complexity due to numerous iterative runs. In contrast, the system proposed in this article relies on (at most) one iteration and, by virtue of using an SVM, does not entail high training complexity. Finally, it is fully unsupervised, meaning that it does not mandate the provision of even a small training set to bootstrap itself.

In research on blocking, the class of Disjunctive Normal Form (DNF) blocking schemes has recently emerged as an important class, with excellent empirical performance on Relational Databases [32, 33, 34]. To the best of our knowledge, it has never been applied to schema-free RDF data. In a preliminary workshop report, we demonstrated that the empirical benefits of relational DNF blocking schemes can also be realized on schema-free RDF data, if the formalism is appropriately adapted [35]. In follow-up work, we showed that the schemes can be used with numerous blocking algorithms, including an adapted version of the classic Sorted Neighborhood algorithm [36]. This article comprehensively develops DNF blocking scheme learning on RDF data, and uses it to propose a novel Set Covering-based learning algorithm with convenient theoretical properties (Section 4.5.1).

2.2. Unsupervised Systems

Among unsupervised systems, existing approaches are mainly based on iterative algorithms that attempt to optimize a manually crafted function called a pseudo f-score [37, 38]. Intuitively, the pseudo f-score is designed to be an accurate reflection of the true f-score. In a recently published study, Ngomo and Lyko showed that pseudo f-scores are not well correlated with actual f-scores [38]. The genetic approach also requires the setting and tuning of many parameters, which may not always be feasible. Finally, the iterative approach proves to be expensive (see Section 6) for all but the smallest datasets, even with blocking.

A second class of approaches rely on Locality Sensitive Hashing (LSH) for computation of features [39], [40], as well as variants of classic unsupervised clustering algorithms such as k-Means and Expectation Maximization (EM) [41]. It has been observed that these approaches tend to work well only if the data is well-structured and meets some stringent conditions [41, 6]. Section 5 evaluates an alternate baseline based on LSH and EM on schema-free, real-world RDF data, with the results showing that these generic algorithms do not perform well generally, especially on noisy text cases.

In contrast to the outlined unsupervised approaches, the proposed work sidesteps supervision by automatically generating its own training samples using heuristics. The subsequent algorithms in the pipeline, including the DNF blocking scheme learner and the machine learning classifier, are technically supervised in that they accept the noisy, automatically generated training samples (converted to feature vectors) as arguments.
Note that the features in this paper rely on phonetic functions. Although these functions were shown in an evaluation to deliver good empirical results at low cost [14], they have not been used as features in any recent state-of-the-art instance matchers. The experiments show that the proposed learning algorithms (and the proposed features) are able to achieve good performance even when the training data contains many incorrectly labeled samples. The system is shown to require at most one extra partial iteration to achieve at-par performance with supervised baselines on several test cases.

Unsupervised alternatives to DNF blocking have also been explored. An example of non-DNF blocking approach that has recently emerged as a viable candidate for blocking schema-free RDF data is Attribute Clustering [13]. This approach is used as a baseline when evaluating the proposed DNF blocking scheme learner.

2.3. Individual Components

Note that some of the individual steps in the proposed architecture have been recognized as important in their own right, and have found additional use-cases. The Dumas schema matcher, for example, was one of the first to use a heuristically generated training set, which resembles (at a high-level) the training set generator used in this paper [42]. We use Dumas’s training set generator as a baseline for evaluating the proposed approach (Section 5.3.1). Recently, we proposed a training set generator for structurally homogeneous Relational Databases, and showed that it could be used for performing unsupervised blocking [34]. In contrast, the generator in this article makes no assumption about structural homogeneity but can process schema-free RDF data. Also, the generated training samples in this article are used for learning both the blocking scheme and the link specification function.

Automatic property alignment has also witnessed recent interest [43]. The Raven system, which learns link specifications based on active learning, was one of the first instance matchers to recognize the need for automatic property alignment [26]. The aligner in Raven matches properties by considering the degree of overlap between the two properties’ object values [26], and casting the property matching as an instance of the stable marriage problem [44]. Another recent work tackled the opposite problem where the instances had already been matched, and the primary task was to match properties [43]. This system also relied on extensional overlap. In the experiments herein, the baseline employed for evaluating the property alignment component is modeled after the overlap principles used by current state-of-the-art property aligners in recent instance matching systems.

A related problem is that of type inference, which attempts to automatically infer the class types of instances by using a form of hierarchical clustering. The state-of-the-art system for this approach is TYPifier [45]. The authors also published a follow-up work where the TYPifier system is used in an instance matcher called TYPiMatch to improve results [46]. The proposed system does not directly perform type inference, but is experimentally shown to successfully handle multi-class instances in a single file. We note that a type inference system can be integrated into the blocking module in Figure 2, if deemed fit by a practitioner.

Finally, note that this article proposes a schema-free instance matcher. Many of the instance matchers in the current literature do not expect schema-free data, but rely on ontological evidence before undertaking instance matching. An example is the KnoFuss system [47]. Other systems rely on the graph-based nature of the RDF data model to directly apply (with minimal modifications) ontology matching techniques to instance matching, examples being ASMOV [48], RiMOM [49] and CODI [50]. More recent efforts attempt to match both ontologies and instances together by cross-fertilizing results, an example being PARIS [51].

These efforts are important but largely orthogonal to the similarity techniques underlying schema-free instance matching, which is not predicated on the existence of metadata. Experimentally, the proposed system is shown to achieve promising f-scores even on test cases where the main source of heterogeneity is semantic (see Section 6), or due to noise at the ontology level. A good example of a recent effort that attempts schema-free instance matching is the system by Rong et al. [52]. We argue that schema-free instance matching is more appropriate for use-cases such as the Web of Linked Data, which has already been dubbed a highly heterogeneous information space in prior work [13, 12].

3. Preliminaries

This article assumes that input datasets are described using the Resource Description Framework (RDF) data model. RDF can be formulated as a set of triples, with a triple defined as:

**Definition 1.** Given a set $B_I$ of blank node identifiers, a set $I_1$ of Information Resource Identifiers (IRIs) and a set $L_V$ of literal values, an RDF triple is a three-element tuple of the form $(subject, property, object)$, where $subject \in B_I \cup I_1$, $property \in I_1$ and $object \in B_I \cup I_1 \cup L_V$.

Note that property is also denoted as predicate by some authors. A triple may be interpreted as a directed, labeled edge, with property indicating the label of the edge going from a subject node to an object node. Thus, an RDF dataset can also be represented as a directed, labeled graph.

The RDF data model is the fundamental data model that underlies Linked Data, defined as a set of best practices for publishing and connecting structured data on the Web [53]. The technology stack that was used to implement this vision comprised RDF, the frequent use of Uniform Resource Identifiers (URIs) and HTTP for dereferencing RDF URIs. As a movement, Linked Data has enjoyed enormous popularity in the last eight years, with Linked Open Data (LOD) currently containing over a thousand datasets and billions of triples [12].

In this article, we assume that all IRIs in the input graphs are URIs, and that the graphs do not contain blank nodes. Both of

---

2 An IRI is more general than a URI.

3 Hypertext Transfer Protocol.
these assumptions are standard in the instance matching community [25]. With these assumptions in place, we define the entity set of an RDF graph as:

**Definition 2.** Given an RDF graph $G$ represented as a set of triples, the entity set $E(G)$ of $G$ is defined as the set of all URIs that occur as a subject in at least one triple in $G$.

The notion of an instance matching link specification function is made precise, before defining the instance matching problem itself:

**Definition 3.** Given two RDF graphs $G_1$ and $G_2$, define an instance matching link specification function as a boolean function $L : E(G_1) \times E(G_2) \rightarrow \{\text{True, False}\}$ that returns True for a pair of entities $(a_1, a_2) \in E(G_1) \times E(G_2)$ iff $a_1$ and $a_2$ refer to the same underlying entity, and False otherwise.

**Definition 4.** Given two RDF graphs $G_1$ and $G_2$ represented as sets of triples, instance matching is the problem of locating all pairs of the form $(a_1, a_2) \in E(G_1) \times E(G_2)$, where $a_1$ and $a_2$ refer to the same underlying entity.

Definition 4 indicates that the instance matching problem is the same as resolving equivalent entities in the two entity sets. For this reason, the instance matching problem is also denoted as entity resolution by some authors [9]. Note also that some authors prefer to denote the range of a link specification function as $[+1, -1]$ instead of True and False, or to assign a specific form to the specification function [54]. In contrast, Definition 3 is designed to be general enough to accommodate the various flavors of link specification functions currently found in the literature. Definition 3 can also be extended to define a probabilistic link specification function that returns a value in the range $[0, 1]$ instead of True or False. We forgo stating this trivially extended definition here, but the notion will prove to be useful in Section 4.5, where an SVM classifier is trained to serve as a probabilistic link specification function.

As described in Section 2, many existing tools assume that the link specification function $L$ is known [10], [21]. This article covers an unsupervised learning version of the problem where $L$ is unknown.

Even if $L$ is known, a naïve approach would evaluate it over the entire Cartesian product $E(G_1) \times E(G_2)$. The blocking step in a two-step instance matcher alleviates this by generating a candidate set of pairs, which is a small subset of the full Cartesian product. It is only over this candidate set that $L$ is evaluated. Intuitively, blocking yields savings without sacrificing coverage because $E(G_1) \times E(G_2)$ is typically sparse in duplicates.

Traditionally, the blocking step uses a function called a blocking scheme in an algorithm called a blocking method [11]. A blocking scheme can be broadly defined as:

**Definition 5.** Given two RDF graphs $G_1$ and $G_2$, define a blocking scheme $\mathcal{B}$ as a boolean function $\mathcal{B} : E(G_1) \times E(G_2) \rightarrow \{\text{True, False}\}$.

Similarly, a blocking method can be defined as:

**Definition 6.** Given two RDF graphs $G_1$ and $G_2$ and a blocking scheme $\mathcal{B}$, define a blocking method as an algorithm that takes $G_1$, $G_2$ and $\mathcal{B}$ as arguments, and outputs a candidate set $\Gamma \subseteq E(G_1) \times E(G_2)$.

According to Definition 5, a blocking scheme takes a pair of entities as input. A non-trivial issue is how a blocking method can efficiently (that is, in near-linear time) use the blocking scheme on the graphs to generate a candidate set. Existing blocking methods work around this issue by explicitly using a blocking key, instead of a blocking scheme.

**Definition 7.** Given two RDF graphs $G_1$ and $G_2$, a blocking scheme $\mathcal{B}$, and an alphabet $\Sigma$, define a blocking key as a pair $(f_1, f_2)$ of many-many functions, where each function $f_i$ has domain $E(G_i)$ and range $\Sigma^*$, and with the constraint that for all entity pairs $(e_1, e_2) \in E(G_1) \times E(G_2)$, $\mathcal{B}(e_1, e_2)$ returns True iff $f_1(e_1) \cap f_2(e_2)$ is non-empty.

The definition above can be intuitively interpreted as follows. Given an entity $e$ from a graph $G_i (i = 1, 2)$, the blocking key applies the function $f_i$ on $e$ and obtains a set of blocking key values (BKV), with a BKV being a string over some alphabet $\Sigma$. An entity can then be thought of as being assigned to multiple blocks, with each block uniquely identified by a BKV. A blocking method individually processes the entities in $E(G_1)$ and $E(G_2)$, and assigns each entity a set of BKVs. Two entities (with one from each graph) can only be paired and added to the candidate set of pairs $\Gamma$ if they share a block; hence, the intersection constraint in Definition 7.

Similar to the earlier definition of the link specification function, Definition 7 is meant to encompass different flavors of blocking keys currently in the literature. Some blocking methods constrain the blocking keys even further, in order to achieve some computational guarantees. The Sorted Neighborhood method, for example, requires that $f_1 = f_2$, and also that $f_i$ is a proper (many-one) function, rather than a relation [7].

Drawing on the preliminaries above, a typical two-step instance matcher can be summarized as a system that first generates a candidate set $\Gamma$ in the blocking step using a blocking key and method; the link specification function $L$ is then evaluated on each pair in $\Gamma$ in the classification step. The differences between individual systems arise in the specific techniques used, as well as the required degree of supervision (see Section 2).

### 4. System

The architecture of the proposed system was earlier illustrated in Figure 2, and briefly described in Section 1. Individual sections below detail the specifics of each component.

#### 4.1. Tabular Serialization of an RDF Graph

As a first step, the input RDF graphs $G_1$ and $G_2$ need to be appropriately serialized. An expressive serialization is important in this problem domain mainly for the efficient processing of entities, as will be described later. In this section, a novel...
logical data structure called a property table is presented that admits representing an RDF graph as a table.

For an RDF graph \( G \) represented as a set of triples, define the property set \( P(G) \) of \( G \) as:

**Definition 8.** Given an RDF graph \( G \) represented as a set of triples, define the property set \( P(G) \) of \( G \) as the set of all URIs that occur as a property in at least one triple in \( G \).

Definition 8 is similar to the definition of a graph’s entity set \( E(G) \), earlier provided in Definition 2. Using the property set, a property schema can be defined by assigning a name to the schema, and using \( \mathcal{A} = P(G) \cup \{ \text{subject} \} \) as the schema’s attribute set \( \mathcal{A} \). The role of the subject attribute will be explained shortly. Figure 3 shows the property table representations of the RDF datasets in Figure 1.

In Figure 3, a property table \( P \) (corresponding to graph \( G \)) is populated by assigning each entity \( e \in E(G) \) its own tuple in the table. Specifically, \( e \) would be an attribute value for the subject attribute in exactly one tuple. This also ensures that the subject attribute is a key for the table. The remaining attribute values get assigned to that tuple based on the object values of the entity \( e \) for the property corresponding to the attribute. Note also that the keyword null and the delimiter ; are both reserved, with their usage demonstrated in Figure 3. Specifically, null is used to indicate that, for a given property, an entity \( e \) has no corresponding object value, while the delimiter ; indicates that \( e \) has multiple object values for the property. The table does not distinguish between object and datatype properties, in keeping with a schema-free representation.

The logical property table is information-preserving since the original graph \( G \) can be reconstructed from it in a straightforward manner. Technically, the described property table is a 1-path property table with attribute set \( P(G) \cup \{ \text{subject} \} \), because each tuple (describing an entity \( e \)) only considers object values of the entity’s immediate properties. Intuitively, the property table is being built by only exploring paths of unit length. Note that it is also possible to generalize this notion to an \( n \)-path property table with an \( n \)-path property schema consisting of the attribute set \( P^n(G) \cup \{ \text{subject} \} \). The table is populated by traversing paths in the graph of (up to) length \( n \). The \( n \)-path property table (for \( n > 1 \)) is also information-preserving, but clearly redundant; the storage of redundant path-based features essentially characterizes a time-space tradeoff. For \( n > 1 \), an \( n \)-path table can be constructed by recursively supplementing the (\( n-1 \))-path table with additional columns. For example, consider Property Table 1 in Figure 3, which is a 1-path table. The 2-path version can be built from this table by supplementing its columns with additional columns. An example of one such supplemented column would be \( d1:\text{hasWife}-d1:\text{hasBrother} \). For the entity \( d1:\text{Mike}_\text{Bates} \), the corresponding value for this column would be \( d1:\text{Sam}_\text{Crax}; d1:\text{Roger}_\text{Crax} \). This value can algorithmically be derived by recursively traversing row entries in Property Table 1, or intuitively by traversing paths of length 2 in the underlying RDF Graph 1 in Figure 1. Note that for all other entities, the value for this column would be null.

The exponential dependence (in the worst case) of the number of columns on \( n \) and the expected sparsity (as the example above illustrates) indicates that for real-world cases, a 1-path property table efficiently represents an RDF graph as a table. The rest of this article assumes that the RDF graph is serialized as a 1-path property table (henceforth denoted simply as property table), although the relevant algorithms are also applicable to \( n \)-path property tables, which would typically be good representations for dense RDF graphs.

There are two reasons for serializing an RDF graph as a property table. First, the table provides an entity-centric characterization of the RDF graph, which allows the algorithms presented in later sections to be expressed in brief, intuitive pseudocode. In part, this is because, when processing two instances, the comparison has to be conducted in terms of the entity’s extensional values, which is captured in a single tuple in the property table. The second reason is that the property table has already been implemented as a physical data structure in triplestores such as Jena [56]. Such triplestores often rely on Relational back-end infrastructure for querying dynamic RDF data. Recent systems, such as UltraVrnap, prove that RDBMS\(^7\) query optimizers can be used to optimize queries in the SPARQL\(^8\) language [57]. A logical adaptation of this physical data structure implies that existing data in triplestores can be batch-processed by the instance matcher in situ, and do not need to be re-serialized.

A straightforward two-pass procedure can serialize an RDF graph \( G \) as a property table in near-linear time in the total number of triples in \( G \). In a first pass over \( G \), the property and entity sets, \( P(G) \) and \( E(G) \), are respectively computed. The property table (with attribute set \( \mathcal{A} = P(G) \cup \{ \text{subject} \} \)) is then initialized, with the subject column populated using \( E(G) \). An index to the subject column is built, with subject attribute values as keys and the corresponding tuple position as value. In a second pass over the triples, the table cells are incrementally updated with each encountered triple. The index ensures the near-linear time guarantee.

4.2 Training Set Generation

Serialized as property tables, the RDF datasets are first input to a training set generator (TSG) that is designed to exploit some inexpensive heuristics to generate both positive and negative training samples. These samples are required in order to train classifiers in subsequent steps.

An effective TSG must overcome at least two challenges. First, the TSG must yield reasonable results without being too expensive, otherwise it risks becoming the computational bottleneck in the full system. In practice, the run-time of an appropriate TSG should be near-linear, similar to other preprocessing steps such as blocking. The second challenge is that of the generated training set quality. Since the TSG relies on heuristics, at

---

\(^6\)This tradeoff was originally demonstrated, not for instance matching, but speedier processing of graph pattern-matching queries (in the SPARQL language) with up to \( n \) self-joins [55].

\(^7\)Relational Database Management Systems.

\(^8\)http://www.w3.org/TR/rdf-sparql-query/
least some fraction of the training set will be noisy, and training set precision falls rapidly as a function of recall with respect to the ground-truth. Prior results on TSGs (verified also by the experiments in this article; see Section 5.3.1) have demonstrated this ‘fall’ to start occurring at relatively low levels of recall [42], [34]. Intuitively, this means that a high-quality training set risks not being representative enough, which could lead to overfitting problems when training classifiers. Conversely, the more representative the set, the noisier it is likely to be.

Algorithm 1 Training Set Generator

| Input : Property tables $P_1$ and $P_2$  
| Parameters $n$ and $\text{thresh}$  
| Tokenizer $T$  
| Output : Set $D$ of positive training samples  
| Set $N$ of negative training samples  
| Method :  
| 1. Initialize empty list $D_1$  
| 2. Initialize empty sets $D$ and $N$  
| 3. Treat each tuple in $P_1$ and $P_2$ as a bag-of-words document by using $T$ to tokenize each tuple  
| 4. Collect term frequency and inverse document frequencies over all documents  
| 5. Collect all tuple pairs $(r, s)$ with logTFIDF score above $\text{thresh}$ in $D'$, where $r \in P_1$, $s \in P_2$  
| 6. Compute Token-Jaccard scores of all pairs in $D'$  
| 7. Sort $D'$ in descending order based on scores computed in previous step  
| 8. Place in $D$ the top $\min(|D'|, n)$ pairs in $D'$, such that a tuple occurs at most once in any pair in $D'$  
| 9. Permute pairs in $D$ to get $N$ distinct pairs, such that $|N| = n$ and $N \cap D$ is non-empty  
| 10. Output $D$ and $N$

This section presents a TSG that was designed keeping these two challenges in mind. The pseudocode of the proposed TSG is provided in Algorithm 1. The TSG tokenizes each tuple in a property table using a tokenizer $T$, and converts it into a bag-of-words document. Drawing on standard information retrieval techniques [58], term frequencies (TF) and inverse document frequencies (IDF) of tokens are computed. The tokenizer $T$ in this article is designed to specifically handle the delimiters often encountered in URIs and other RDF elements. In preliminary experiments, off-the-shelf tokenizers were found to be inadequate for the challenges (such as URI prefixes) posed by RDF elements in linked datasets.

The logTFIDF score (equivalently the logarithm of the cosine similarity score, as denoted by information retrieval practitioners) is given by the formula below:

$$\log \text{TFIDF}(r, s) = \sum_{q \in r \cap s} w(r, q) \cdot w(s, q)$$  

where, for any tuple $t$ and term $q$,

$$w(t, q) = \frac{\sum_{r \in t} w'(t, q)}{\sqrt{\sum_{s \in t} w'(t, q)^2}}$$

and where,

$$w'(t, q) = \log(tf_{t, q} + 1). \log \left( \frac{|P|}{df_q} + 1 \right)$$

The equations assume that $r$ and $s$ are tuples from property tables $P_1$ and $P_2$ respectively, $w(t, q)$ is the normalized TFIDF weight of a term $q$ in a tuple $t$ (from either property table), $tf_{t, q}$ is the term frequency of $q$ in $t$, $|P| = |P_1| + |P_2|$ is the total number of tuples in both property tables and $df_q$ is the number of tuples in which the term $q$ appears. Note that IDF statistics are collected over both property tables.

Using the parameter $\text{thresh}$ as a filter, only the pairs with logTFIDF score above $\text{thresh}$ are retained in the list $D_1$. If $\text{thresh}$ is too high, there may be fewer than $n$ pairs with score above $\text{thresh}$. In practice, setting $\text{thresh}$ to a default low value (such as 0.001) is found to suffice (Section 5.3.1). The rationale behind setting a low (but non-zero) $\text{thresh}$ is to eliminate the vast majority of pairs that only have unimportant tokens (such as http) in common. A default value of $\text{thresh}$ can be set in a self-tuning manner in an actual implementation; if fewer than $n$ samples are returned, $\text{thresh}$ decreases by a small value till $n$ samples are returned by Algorithm 1.
Efficient implementations of the logTFIDF function have been extensively researched in the information retrieval community and drawing on the prior work of Cohen [59], we implemented lines 1-5 of Algorithm 1 with guaranteed runtime $O(\alpha(|P| ||A_1| + |P_2||A_2|))$ where $\alpha$ is a slow-growth (near-constant) function, and $A_i$ is the attribute set of table $P_i$.

The tuple pairs collected in $D'$ are scored in line 6 using the Token-Jaccard similarity measure. Given two token-sets $S_1$ and $S_2$ as input, their Token-Jaccard score is given by:

$$Token - Jaccard(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

A key property of Token-Jaccard is that it is a local similarity function in that it does not rely on statistics such as IDF that require a pass over the entire dataset. Since logTFIDF and threshold have already served as a filter for eliminating obvious non-duplicates, Token-Jaccard can be used to further refine and sort $D'$. In line 8, the top $n$ (or $|D'|$, whichever is smaller) pairs in the sorted list are added to the output set $D$.

A natural question is if the Token-Jaccard refinement step is even necessary, given that logTFIDF is roughly accomplishing the same goals. In fact, the training set generator used by the Dumas schema matcher does not bother with this step, but sorts the list based on the logTFIDF scores and outputs the top $n$ results [42]. The rationale for including this step is that Token-Jaccard places higher emphasis on token overlap with respect to the union of the tokens-sets, and is agnostic to how common the tokens are in the other tuples. This aggressive strategy would be necessary to obtain non-duplicates, but logTFIDF already filtered out non-duplicates with high token overlap. The benefits of using two heuristics instead of one will be empirically demonstrated in Section 5.3.1.

Another constraint that should be noted in line 8 is that a tuple (from either property table) occurs at most once in $D'$. Intuitively, this constraint attempts to make the training sets as representative as possible by preventing a single tuple from getting undue coverage in the training set. This relates directly to the quality-representation tradeoff earlier mentioned as a challenge in prior TSG work [42], [34].

**Example 1.** Suppose $n = 3$ and the sorted list at the end of line 7 in Algorithm 1 is $D' = [(r_1, s_1), (r_2, s_3), (r_1, s_7), (r_5, s_1)]$, where $r_i$ and $s_j$ denote the $i^{th}$ and $j^{th}$ tuples in property tables $P_1$ and $P_2$ respectively. The chosen positive training set would then be $D = [(r_1, s_1), (r_2, s_3), (r_6, s_3), (r_1, s_7)]$, since tuple $r_1$ has already appeared in a higher-scoring pair.

Non-duplicates can be automatically generated by relying on the observation that real-world datasets are often sparse in duplicates. This assumption is also predicated by the blocking step, which can only be applied if the vast majority of pairs are assumed to be non-duplicates [11]. Line 9 in Algorithm 1 permutes the pairs in $D$ to obtain new pairs ($\mathcal{N} \setminus D$) that are assumed to be non-duplicates. To achieve balanced training, $|\mathcal{N}| = n$.

**Example 2.** Continuing from the previous example, where $n = 3$ and the generated duplicates-set was $D = [(r_1, s_1), (r_2, s_3), (r_6, s_3), (r_1, s_7)]$, an example of $N$ generated by permuting $D$ would be $[(r_6, s_3), (r_1, s_1), (r_2, s_3)]$.

In practice, such a permutation is found to lead to near-perfect accuracy on the generated non-duplicates set (Section 5.3.1). Finally, note that while lines 1-8 of Algorithm 1 (and by virtue, $D$) are deterministic, there are usually many possibilities for $N$. An alternative option for generating $N$ is to randomly pair tuples in $P_1$ with tuples in $P_2$. Aside from the additional computational cost incurred by such an option, the adopted approach is expected to exhibit less randomness, since both sets $D$ and $N$ are constructed using common tuples$^9$ and $n$ is expected to be small compared to dataset sizes.

### 4.3. Property Alignment

This section presents an algorithm for performing alignment between the attribute sets $A_1$ and $A_2$ of the two property tables $P_1$ and $P_2$ respectively. The alignment algorithm only uses the training samples that were generated by Algorithm 1, and not the full datasets.

#### Algorithm 2 Property Aligner

**Input :** Sets $D$ and $N$ of positive and negative training samples respectively

**Attribute sets $A_1$ and $A_2$**

**Output :** Property Alignment Set $Q$

**Method :**

1. Initialize empty set $Q$
2. Initialize numeric variable $avg := 0.0$
3. Initialize empty $|A_1| \times |A_2|$ dimensional matrix $M$
4. for all attribute pairs $(a_1^i, a_2^j) \in A_1 \times A_2$
   
   if URI stems of $a_1^i$ and $a_2^j$ exactly match then
   
   Add $(a_1^i, a_2^j)$ to $Q$
   
   end if
   
5. end for
6. for all Attribute pairs $(a_1^i, a_2^j) \in A_1 \times A_2$
   
   $M[i, j] := ColumnSimim(D, a_1^i, a_2^j) - ColumnSimim(N, a_1^i, a_2^j)/|Q|$
   
7. end for
8. for all pairs $(a_1^i, a_2^j) \in Q$
   
   Let $avg := avg + (ColumnSimim(D, a_1^i, a_2^j) - ColumnSimim(N, a_1^i, a_2^j))/|Q|$
   
9. end for
10. for all entries in $M$ do
    
    if entry $M[i, j] \geq avg$ then
    
    Add the pair $(a_1^i, a_2^j)$ to $Q$
    
    end if
    
11. end for
12. Output $Q$

$^9$The probability of picking a non-duplicate tuple pair by randomly picking a pair from $P_1$ and $P_2$ is (assuming duplicates-sparseity) approximately $1/(|P_1||P_2|)$, whereas for the adopted approach, it is only of the order $1/(n^2 - n)$. 

8
For notational succinctness, refer to an attribute of \( A_i \) as \( a_i^1 \) and an attribute of \( A_j \) as \( a_j^2 \) where \( i \) and \( j \) respectively range from 1 to the number of attributes in \( A_1 \) and \( A_2 \). Using this notation, a property alignment is simply defined as an element of the set \( A_1 \times A_2 \). Let \( Q \) denote a property alignment set. If the attribute sets of the input property tables are interpreted in a manner similar to Relational Database (RDB) schemas, the alignment set \( Q \) would be like the set generated by a schema matcher with local 1:1 cardinality but global \( m:n \) cardinality. A survey of RDB schema matching was provided by Rahm and Bernstein [60]. The following example illustrates the concept.

**Example 3.** Consider the two property tables in Figure 3. The property alignment set \( Q \) should ideally contain the alignments \((Subject, Subject), (d1:hasWife, d2:spouse), (d1:hasBrother, d2:sibling), (d1:hasBrotherInLaw, d2:inlaw), (d1:year, d2:birthdate), (d1:month, d2:birthdate), (d1:day, d2:birthdate)\), since alignments can be partial. The global cardinality is \( m:n \) since an attribute participates in more than one alignment in \( Q \). The local cardinality is 1:1 since each alignment is between two attributes and not two sets of attributes.

In this article, a property alignment \((a_i^1, a_j^2)\) is meant to indicate a (possibly partial) match between the \( i^{th} \) and \( j^{th} \) columns of \( P_1 \) and \( P_2 \) respectively. The alignment set \( Q \) is not important by itself, but like the training set, will prove to be an important input to the subsequent feature generator component. Intuitively, the alignment set will enable the feature generator to constrain the size of the feature space (Section 4.4).

One solution to generating an alignment set is to use an instance-based matcher such as Dumas [42]. Dumas, as earlier stated, generates noisy duplicates using its own TSG and performs (both global and local) 1:1 schema matching. As the simple example of Figure 3 shows, global 1:1 alignments are not adequate for this task and could lead to loss of information.

Secondly, Dumas does not consider the names of attributes, which can be quite indicative, especially in Linked Data property namespaces [61]. As a simple application of this finding, consider the fact that the first column of every property table is always named \( subject \); the alignment (\( subject, subject \)) should always be included in \( Q \). At the same time, the \( birthdate \) column is a global \( m:n \) element since it is not a URI; it is assumed to be its own URI stem. The alignment \((subject, birthdate)\) in Example 3 shows that the names can be problematic, since the property \( d1:date_of_birth \) is lexically more similar to \( d2:birthdate \) than \( d1:year, d1:month \) and \( d1:day \).

Similar issues arise if column-based matchers are adapted instead of instance-level matchers (e.g., Dumas). Column-based matchers match columns based on the degree of overlap between their value-sets. Several property aligners used both within and without the context of instance matching employ a similar technique based on extensional (or object-value overlap) of RDF properties. Some of these were described in Section 2. Both Dumas and a generic column-based matcher are used as baselines when evaluating the property alignment step in Section 5.3.2.

To address the described challenges, a hybrid parameter-free property aligner is proposed. The aligner considers both the names of the properties, as well as columnar aggregations of training data. Algorithm 2 shows the pseudocode of the property aligner. First, the algorithm strips the URI prefixes of property columns and uses a basic exact-match indexing procedure on the resulting URI stems (after converting them to lowercase strings) to heuristically determine the trivial 1:1 alignments (lines 1-5). An obvious consequence is that \( Q \) is guaranteed to include at least the alignment \((subject, subject)\).

Before describing the rest of the algorithm, the ColumnSim score over a set \((say \ D)\) of \( n \) tuple pairs \( D = \{(r_1, s_1), \ldots, (r_n, s_n)\}\) is computed as follows. The ColumnSim function takes as input \( D \) and two attributes, \( a_i^1 \in A_1 \) and \( a_j^2 \in A_2 \). Denote as \( R \) and \( S \) the tables containing the tuples \( r_1, \ldots, r_n \) and \( s_1, \ldots, s_n \) respectively. ColumnSim tokenizes the \( i^{th} \) and \( j^{th} \) columns (using the same tokenizer \( T \) in Algorithm 1) of \( R \) and \( S \) respectively to obtain two sets of tokens \( R' \) and \( S' \). The Token-Jaccard score (Equation 4) of \( R' \) and \( S' \) yields the final ColumnSim score.

In lines 6-7, a matrix \( M \) is populated, with the \((i, j)^{th} \) cell of the matrix containing the value obtained by subtracting the ColumnSim scores of the corresponding attributes \( a_i \) and \( b_j \) over \( D \) and \( N \). The subtraction serves as a conservative filter to prevent accidental matches from happening. Using the current alignments in \( Q \) (obtained earlier through exact matching of URI stems), the average score of matrix cells corresponding to elements in \( Q \) is computed as avg, and used as an automatic threshold to pick property alignments (line 10). The resulting alignment set \( Q \) is then output (line 12).

Using a hash-based method, the loop in line 4 run in time \( O(|A_1| + |A_2|) \). Assuming (based on characteristics of commonly encountered real-world data) that within an attribute set \( A \), no two attributes have the same URI stems, \( Q \) (at the end of line 5) has maximum cardinality \( \min(|A_1|, |A_2|) \). Populating the matrix in lines 6 and 7 can be done in time \( O(|D| + |N|)\), making this the most expensive step of the algorithm. The run-time of this step subsumes the computations in the remainder of the algorithm, since lines 8-11 require two passes over the matrix \( M \).

In practice, Algorithm 2 was found to run near-instantaneously even in the case of a benchmark with over a hundred properties (Section 5.3.2). The parameter-free nature of Algorithm 2 lends it an advantage in that it can be run like an off-the-shelf black box by a practitioner, precluding the need for cumbersome parameter tuning. To the best of our knowledge, a hybrid parameter-free property aligner does not exist in the current research literature.

\[10\]In the Raven system, for example, stable matching of properties primarily relied on object-value overlap [26].
4.4. Feature Generator

The training set and property alignments are now input to a feature generator, which converts each tuple pair in the training set to a feature vector, as subsequently described. The output of the feature generator is two sets containing \( n \) feature vectors each, where \( n \) is the number of duplicates (and also non-duplicates) in the training set.

We will continue to use the property tables, \( P_1 \) and \( P_2 \), in Figure 3 as running examples. Recall that the attribute sets of \( P_1 \) and \( P_2 \) were respectively denoted by the symbols \( A_1 \) and \( A_2 \). An attribute in \( A_1 \), representing the \( i^{th} \) column in \( P_1 \), is denoted by the symbol \( a_i \); similarly for an attribute in \( A_2 \). Finally, the symbols \( r \) and \( s \) are again used to denote generic tuples from \( P_1 \) and \( P_2 \) respectively.

Using the symbol \( \ast \) for the Kleene star, we define a property-specific indexing function (P-SIF) as follows:

**Definition 9.** Given an alphabet \( \Sigma \), a property table \( P \), and an attribute \( a_i \) from the attribute set of \( P \), define a property-specific indexing function \( h_i : P \rightarrow 2^\Sigma \) as a function that takes as input a tuple from the table \( P \) and is applied on the attribute value of the tuple corresponding to \( a_i \). The resulting output is a set \( Y \) of strings over the alphabet \( \Sigma \).

While technically possible to construct a special P-SIF \( h_i \) for the \( i^{th} \) column of the table, it is more appropriate for an unsupervised procedure to consider a set \( \mathcal{G} \) of general indexing functions or GIFs. A GIF is a generic function that accepts a string as input and returns a set of strings as output. Given such a property-agnostic set \( \mathcal{G} \) and an attribute set \( A \) of some property table, the set \( \mathcal{H} \) of all possible P-SIFs can be constructed by considering the Cartesian product of \( \mathcal{G} \) and \( A \). If some function in \( \mathcal{G} \) does not apply to an attribute in \( A \), the P-SIF returns the empty set.

**Example 4.** Consider the first tuple of \( P_1 \) in Figure 3 and the simple GIF Tokens, which accepts a string as input, tokenizes it and returns the set of tokens as output. Applied to each of the eight attributes in \( A_1 \), eight P-SIFs \( h_1, \ldots, h_8 \) can be constructed. For the null attribute values, an empty set would be returned. On the other hand, consider a GIF AddOneToIntegers. This GIF would also parse the tokens in the string but it would discard all tokens that are not integers. The tokens that can be parsed to integers would be parsed, incremented, re-converted to strings and output as a set. AddOneToIntegers would only be applicable to certain numeric attributes (such as \( d1 : day \) in \( A_1 \)), and would return the empty set for all others.

In the rest of the article, it is assumed that the sets \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) of P-SIFs are formed over respective attribute sets \( A_1 \) and \( A_2 \), by forming the cross-product of the attribute set with a given set \( \mathcal{G} \) of GIFs. Thus, it is always the case that \( |\mathcal{H}_i| \) equals \( |\mathcal{G}|.|A_i| \). Intuitively, the set \( \mathcal{G} \) forms the atomic feature set, from which feature spaces for each property table are individually constructed by using the attribute set. Before describing this procedure, we describe the twenty-eight GIFs used in this article below.

1. **Identity:** Returns a singleton set containing the string.
2. **Tokens:** Tokenizes the string based on a set of delimiters specifically designed for RDF elements, and outputs the set of tokens.
3. **Integers:** Similar to (2) but discards all strings in the output that cannot be parsed as integers.
4. **ManipulateIntegersByOne:** Same as (3), except that for every integer \( a \), integers \( a-1 \) and \( a+1 \) are converted to strings and added to the output set along with \( a \). Note that the GIF AddOneToIntegers described in Example 4 is a simplified version of this GIF.
5. **ExtractNCharPrefixes:** Same as (2) except that each token is further truncated to its first \( N \) characters. If the token has fewer than \( N \) characters, it is left intact. Three GIFs were implemented, with \( N \) set to 3, 5 and 7 respectively.
6. **ExtractTokenNGrams:** Tokenizes the string as an ordered list and extracts length-\( N \) contiguous subsequences of tokens. If the list of tokens contains fewer than \( N \) tokens, the list becomes its own only subsequence. Each subsequence is added to the output set. Implemented for \( N=2,4,6 \).
7. **ExtractNonSoundexPhoneticFeatures:** Tokenizes the string and adds the phonetic encoding of each token to the output set. The phonetic functions used for implementing seven GIFs in total are Caverphone1, Caverphone2, ColognePhonetic, DoubleMetaphone, MatchRatingApproachEncoder, Metaphone and NYSISI. The popular Soundex encoding is treated specially (see below). A library implementing all these encoding functions efficiently exists in an Apache open-source package\(^{14}\) and was adapted for this article.
8. **ExtractSoundexPhoneticFeatures:** Tokenizes the string and adds the Soundex encoding of each token to the output set. We consider the original Soundex encoding algorithm (implemented in the Apache open-source package), a refined version (also implemented in the package) as well as eight variations implemented in the open-source FEBRL package\(^{29}\). An example of a variation is to truncate each Soundex encoding to only the first four characters.
9. **ExtractAlphaNumeric:** Extracts all tokens from the string such that a token contains at least one alphabet as well as a numerical digit (in addition to other optional characters). A rationale for this feature is subsequently provided.

The first ten GIFs are standard and have already been found to work well in previous work, including the original Relational Database setting in which they were first proposed\(^{32}\). A brief rationale for the features is provided below. We provide further details and accompanying examples on the project website\(^{15}\).

GIFs 1-2 are appropriate for strings that have high token overlap or for alphanumerical codes (in product databases, for example) that tend to match exactly and have high correlation with duplicate classification. GIFs 3-4 are more appropriate for phone numbers, zip codes, street numbers, social security numbers, dates of birth and other numeric quantities that commonly occur in databases. GIFs 5-7 are empirically robust to many

\(^{14}\)org.apache.commons.codec.language

\(^{15}\)https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/unsupervised-im
data representation issues; for example, GIF 5 would not distinguish between strings that spell ‘Avenue’ as Avenue or Ave. GIFs 8-10 generate token N-grams and are useful for detecting discriminative phrases in long descriptions.

In his comprehensive text, Christen evaluates the phonetic encodings, including Soundex and its variations, used by GIFs 11-27 [14]. The advantage of phonetic functions is that they are robust to spelling variations (especially in names) that the other GIFs cannot easily accommodate (e.g. Kathryn vs. Catherine). Using a range of phonetic encodings compensates for the quirks of a single encoding. Variations of phonetic encodings can further help to compensate for other sources of noise, such as missing prefixes and extreme misspellings. Since phonetic encodings are not trivial to compute, it makes computational sense to only consider variations of one particular phonetic encoding. The Soundex encoding was chosen for this purpose because it is well-studied and has an efficient, transparent implementation in packages such as FEBRL [62], [29].

Algorithm 3  Converting Training Set to Sets of Feature Vectors

\textbf{Input :} Training set of duplicates \( D \) and non-duplicates \( N \)

\textbf{Property Alignment Set } \( Q \)

\textbf{Set } \( G \) of General Indexing Functions (GIFs)

\textbf{Output :} Sets \( D_f \) and \( N_f \) of duplicates and non-duplicates feature-vectors respectively

\textbf{Method :}

1. Initialize empty feature-vectors sets \( D_f \) and \( N_f \)
2. \textbf{if} \( Q \) is empty \textbf{then}
   \( Q := \mathcal{A}_1 \times \mathcal{A}_2 \)
3. \textbf{end if}
4. \textbf{for all} tuple pairs \((r,s)\) \textbf{in} \( D \) \textbf{do}
   \textbf{Initialize} \( f \) to a \(|Q||G|\) 0-vector
   \textbf{for all} alignments \((a_1, a_2)\) \textbf{in} \( Q \) \textbf{do}
   \textbf{for all} GIFs \( g \in G \) \textbf{do}
   \textbf{Let} \( h^1 \) and \( h^2 \) be the P-SIFs obtained by combining \( g \) with \( a_1^1 \) and \( a_2^2 \) respectively
   \textbf{Let} \( f_{ij} \) denote the property-specific feature formed from \( h^1 \) and \( h^2 \)
   \textbf{if} \( f_{ij}(r,s) = 1 \) \textbf{then}
   \( f[|G| \times i + j] := 1 \)
   \textbf{end if}
   \textbf{end for}
   \textbf{end for}
   \textbf{Add} \( f \) to \( D_f \)
5. \textbf{end for}
6. \textbf{Repeat} steps 4-5 by iterating over tuple pairs in \( N \), and populate \( N_f \)
7. \textbf{Output} \( D_f \) and \( N_f \)

Finally, the utility of GIF 28 is best realized in the cases where ID strings are often present and can be used to identify duplicate entities. Such strings tend to have both alphabets and digits and are relatively rare. Compared to more general token-based features (such as GIF 2), GIF 28 tends to be more discriminative, which helps the subsequent feature selection process.

Let \( h^1_i \) denote a P-SIF in \( \mathcal{H}_1 \) (and a similar analysis applies to a P-SIF \( h^2_i \) in \( \mathcal{H}_2 \)), where \( h^1_i \) is the P-SIF obtained by combining the GIF \( g \in G \) and the attribute \( a_1^i \in \mathcal{A}_1 \). Applied to a tuple \((r,s)\), let the output of the P-SIF \( h^1_i \) be denoted as \( h^1_i(r) \). Given this notation, let a \textit{property-specific feature} be defined as follows:

\textbf{Definition 10.} Given two P-SIFs \( h^1_i \in \mathcal{H}_1 \) and \( h^2_j \in \mathcal{H}_2 \), define a \textit{property-specific feature} \( f_{ij} \) as a binary function that takes as input a tuple pair \((r,s)\) and returns 1 \textit{iff} \( h^1_i(r) \cap h^2_j(s) \) is non-empty, and returns 0 otherwise.

Given an alignment set \( Q \), each tuple pair in the training set can be converted to a feature vector with \(|G||Q|\) binary elements, with each element corresponding to a single invocation of a property-specific feature \( f_{ij} \) on the tuple pair, where \((a_1^i, a_2^j)\) is an element in \( Q \). The pseudocode is provided in Algorithm 3.

Note that the dimensionality of each feature vector is directly proportional to \( Q \). In the event that \( Q \) is unavailable, the only recourse (a ‘fallback’ option) for the system is to consider the exhaustive set \( \mathcal{A}_1 \times \mathcal{A}_2 \) (line 2). This demonstrates why having a compact, high-recall property alignment set \( Q \) is important, since both the quality and size of the resulting feature space depends on the quality of, and number of alignments in, \( Q \).

Given a training set with \( n \) duplicates and non-duplicates, the feature generator outputs two feature sets with \( n \) binary vectors each. The time taken by Algorithm 3 is \( O(cn|G||Q|) \), assuming that the run-time of each GIF \( g \in G \) can be bounded above by \( O(c) \).

4.5. Learning Procedures

The two sets of feature vectors are now input to two independent training procedures, which respectively learn a blocking scheme (see Definition 5) for the blocking step and an SVM classifier, which serves as a probabilistic link specification function for the classification step.

4.5.1. Blocking Scheme Learner

In the following discussion, let \( Q \) denote the set of precisely those property-specific features that have the value 1 in at least one feature-vector in the set \( D_f \cup N_f \). In other words, \( Q \) contains property-specific features that \textit{cover} at least one feature vector in the training set. An obvious upper bound on \(|Q|\) is \(|G||Q|\), since each feature vector has at most \(|G||Q|\) elements. In practice, the diversity of the twenty eight GIFs in Section 4.4 results in \(|Q|\) being less than \(|G||Q|\). Interpreting each of the features in \( Q \) as a \textit{boolean} variable, a \textit{property-specific blocking scheme} in Disjunctive Normal Form can be defined as follows:

\textbf{Definition 11.} Given a set \( Q \) of property-specific features, define a \textit{property-specific Disjunctive Normal Form blocking scheme} \( B \) as a disjunction of terms, where each term is a conjunction of features from \( Q \).

\[^{16}\text{With the 1 value interpreted as True and 0 as False.}\]
to learn a DNF blocking scheme, k

tive blocking scheme

Q

positive

the class of property-specific DNF blocking schemes defined in

DNF blocking schemes devised for RDBs is a special case of

ing schemes being defined and used for RDF data. The class of

our knowledge, this is the first instance of this class of block-

liver excellent empirical performance [32], [33]. To the best of

homogeneous Relational Databases (RDBs) and found to de-

tings. Let a 1-DNF blocking scheme be denoted as a

disjunc-

true threshold parameter κ

This class of blocking schemes (henceforth, simply referred

to as DNF blocking schemes) was first proposed for structurally

homogeneous Relational Databases (RDBs) and found to de-

liver excellent empirical performance [32], [33]. To the best of

our knowledge, this is the first instance of this class of block-

schemes being defined and used for RDF data. The class of

DNF blocking schemes devised for RDBs is a special case of

the class of property-specific DNF blocking schemes defined in

Definition 11. Note that a DNF blocking scheme is a positive

formula, since a term cannot contain negated features from Q.

Let the DNF blocking scheme be denoted as a k-DNF blocking

scheme if each term is constrained to contain at most k fea-

ures. Let a 1-DNF blocking scheme be denoted as a disjunc-

tive blocking scheme, given that it is merely a clause. In order

to learn a DNF blocking scheme, k must be specified as a pa-

parameter. The DNF blocking scheme is said to cover a tuple pair

(or its equivalent feature vector representation) if it evaluates to

True for that pair. Ideally, the learned scheme should cover as

many of the duplicates as possible, while minimizing coverage

of the non-duplicates.

The problem formulation described above is similar to that

of the classic Set Covering (SC) problem [63]. This connection

(between DNF blocking scheme learning and SC) was first

shown by Bilenko et al., when the DNF learning problem for

structurally homogeneous RDBs was reduced to Red-Blue SC

[32], [64]. Although the problem in this article is more general

(since the property tables are not structurally homogeneous), a

similar reduction applies here.

Unfortunately, SC is NP-Complete\footnote{Most known variants are also NP-Complete [65].} [65]. Using an additional

threshold parameter κ, an SC approximation algorithm from

the literature can be leveraged to learn a k-DNF blocking

scheme. The pseudocode is given in Algorithm 4.

In line 1, Algorithm 4 uses k to supplement the set Q and

obtain the set Q_k. If we define an i-term as a term that is con-

structed by forming a conjunction of exactly i property-specific

features from Q, and S_i as the set of all possible i-terms, Q_k is

\begin{align*}
Q_k &= \bigcup_{i=1}^{k} S_i 
\end{align*}

Note that S_1 is simply the alignment set Q. In practice, not

all terms will be used by Algorithm 4, making an exhaustive

construction of Q_k unnecessary. Instead, a pruning strategy

includes only those terms in Q_k that cover some feature-vector in

D_f \cup N_f, since only those terms will actually be used (lines 2-

3). Figure 4 illustrates the strategy. Assuming that k = 2, there

are three possible 2-terms a AND b, a AND c and b AND c, but

only the two shown in Figure 4 would get added to the supple-

mented set Q_k. If k = 3, then Q_3 = Q_2. This is because there

is no feature-vector that is simultaneously covered by a term

with three property-specific features from Q. Using the sup-

Algorithm 4 Property-specific DNF Blocking Scheme Learner

Input : Sets D_f and N_f of duplicates and non-duplicates

feature-vectors

Set Q of property-specific features

Term parameter k

Set cover threshold parameter κ

Output : Property-specific k-DNF Blocking Scheme B

Method :

1. Supplement set Q to get set Q_k (Equation 5)
2. Construct M_D = \langle X, Q_k \rangle. X is a feature vector in D_f, 

Q_k \subseteq Q_k contains the elements in Q_k covering X
3. Reverse previous step to build M_N for feature vectors in

N_f
4. Reverse M_D and M_N to respectively get M_D’ and M_N’
5. for all X in keyset(M_D’) do

Score X by using formula

\[ |M_D’(X)|/|D_f| - |M_N’(X)|/|N_f| \]

Remove X if score(X) < κ
6. end for
7. Perform Weighted-SC on keys in M_D’ using Chvatal’s

heuristic [63], with weights set to the negation of the

scores calculated above
8. B := disjunction of chosen keys
9. Output B

Figure 5: Construction of multmaps and reversed multmaps in Algorithm 4, assuming the information in Figure 4
implemented set \( Q \), lines 2-3 construct multmaps\(^{18} \) by assigning each feature-vector in \( D_f \) a key in \( M_D \), and with the elements in \( Q \), covering that feature-vector comprising its value set. \( M_D \) is then reversed to yield \( M_D' \). \( M_N' \) is similarly constructed. Figure 5 demonstrates the key-value reversal procedure, assuming \( D_f \) contains feature-vectors 1-5, covered as shown in Figure 4. The time complexity of building (both) \( M_D' \) and \( M_N' \) is \( O(|Q|([|D_f|]+|N_f|)) \).

In lines 5-6, each key is first scored by calculating the difference between the fractions of covered duplicates and non-duplicates. A threshold parameter, \( \kappa \), is used to remove the keys that have low scores. \( \kappa \) is designed to improve quality by removing those features from \( Q \) that either cover too few duplicates, or cover too many non-duplicates (or both). The range of \( \kappa \) is [-1,1]. A value close to 1.0 would indicate that the user is confident about low noise-levels in inputs \( D, N \) and the property alignment set \( Q \), since high \( \kappa \) implies the existence of elements in \( Q \) that cover many positives and few negatives. Since many keys in \( M_D' \) are removed by high \( \kappa \), this also leads to computational savings. However, setting \( \kappa \) too high (perhaps because of misguided user confidence) could lead to excessive purging of \( M_D' \), and subsequent failure of Algorithm 4. A low \( \kappa \) is safer, but may result in slower run-times.

Similar to the parameter \( \text{thresh} \) in Algorithm 1, \( \kappa \) can also be set in self-tuning mode, with a low (but not too low) default value of 0.2. If Algorithm 4 fails with a given value of \( \kappa \), it is indicative of \( \kappa \) being too high. \( \kappa \) is then decreased by a small number (e.g. 0.05) till Algorithm 4 successfully returns a blocking scheme. In one of the conducted experiments (Section 5.3.3), the self-tuning methodology is found to lead to seamless execution of Algorithm 4.

In line 7, Weighted Set Covering (W-SC) is performed using Chvatal’s approximation algorithm \([63]\), with each key in \( M_D' \) acting as a set and the tuple pairs covered by all keys as elements of the universe set \( \mathcal{U} \). For example, assuming that all features in \( Q \), in the keyset of \( M_D' \), in Figure 5 have scores above \( \kappa \), \( \mathcal{U} = \{1, 2, 3, 4, 5\} \). Note that only \( M_D' \), is pruned (using \( \kappa \)) and also, W-SC is performed only on \( M_D'. \) \( M_N' \) only aids in the score calculation (and subsequent pruning process) in line 5 and may be safely purged from memory before line 7.

W-SC needs to find a subset of the \( M_D' \) keyset that covers all of \( \mathcal{U} \) and with minimum total weight. For this reason, the weight of each set is the negative of its calculated score. Given that sets chosen by W-SC actually represent features in \( Q \), their disjunction is the desired k-DNF blocking scheme (line 8).

As stated before, Set Covering (and also Weighted Set Covering) is known to be an NP-Complete problem \([65]\). Under plausible\(^{19} \) complexity assumptions, Chvatal’s algorithm is currently the best-known polynomial-time approximation for W-SC \([66]\). Since Algorithm 4 directly invokes Chvatal’s algorithm as a subroutine, it is conferred with similar theoretical guarantees. In practice, setting \( k \) to 1 has been shown to be a viable option even on noisy test cases \([67]\). This is an important computational benefit since \( |Q| \) is exponential in \( k \) in the worst-case.

4.5.2. Training the Classifier

The feature-vectors sets \( D_f \) and \( N_f \) are also used for training a supervised classifier that serves as a probabilistic link specification function in the classification step. Note that although we re-use the sets \( D_f \) and \( N_f \) for training the classifier (Figure 2), it is theoretically possible to devise a new feature space for this step. For example, one could add a new floating-point valued feature \( \text{Levenstein} \), yielding \( |Q| \) new features\(^{20} \) for each tuple pair in \( D \) and \( N \). Indeed, a similar supplemental step was performed in Algorithm 4 for learning k-DNF blocking schemes, when \( k > 1 \). In this article, we continue to use the binary feature-vectors output by Algorithm 3 for training the classifier.

The primary reason for re-using the original feature-vectors is computational. Each additional feature computation incurs cost \( |Q| \) for each tuple pair, and would additionally increase the run-time for training a classifier. Re-using the feature vectors further implies that the feature generator only needs to be run once and that in a shared-memory architecture, both the DNF blocking scheme learner and the classifier trainer can access the same feature-vectors, resulting in savings in both time and space.

As for the specific classifier trained on the feature-vectors, we note that the noise in the training sets, the sparsity of non-zero elements in individual feature-vectors and the potential curse-of-dimensionality issue that would arise if \( Q \) is large compared to either \( D \) or \( N \), all indicate the use of a kernel-based SVM classifier \([68]\). Previous studies have validated this empirically by showing that supervised SVM-based classifiers such as FEBRL and MARLIN achieve state-of-the-art performance on standard benchmarks \([29, 28]\). An interesting issue that we evaluate in this paper is whether a kernelized SVM trained on noisy samples also demonstrates similar benefits.

In this article, we use the training sets \( D_f \) and \( N_f \) to train an SVM with a Radial Basis Function (RBF) kernel \([69]\). We do not adopt a polynomial kernel because it requires the tuning of more hyper-parameters, which is problematic given that the system only has a limited, noisy number of training samples available to it. It is also known that a linear kernel (and for certain parameters, a sigmoid kernel) is a special case of the RBF kernel, making it a reasonable choice \([70]\).

Finally, while more sophisticated machine learning classifiers can always be used in this module instead of a kernelized SVM, a user should be aware of their typically higher training times. For example, multilayer perceptrons, which were recently shown to deliver slightly better performance on average than SVMs, were simultaneously found to be almost an order of magnitude slower on several test cases \([27]\).

4.6. Blocking Method and Second Step

Given a blocking key (Definition 7), there has been extensive research on how best to use the key in a blocking method.
including a variety of methods specifically designed for heterogeneous information spaces such as the Web of Linked Data [13]. A promising blocking method is block purging. The method works by using a given blocking key on each entity to generate blocking key values (BKV). Entities are clustered into (possibly overlapping) blocks, with each block uniquely identified by a BKV. To control data skew, block purging eliminates all blocks that generate more pairs than a threshold, designated in this paper as maxPairs. An algorithm was proposed to calculate maxPairs automatically, but required a two-pass approach over the generated blocks [13]. In preliminary experiments, we found that manually determining the maxPairs threshold led to significantly superior results over automatically determining maxPairs. Tuning maxPairs was also not found to be cumbersome; hence, this approach is adopted in the current implementation. Since the blocking method is a separate module (Figure 2), practitioners can customize this step per their needs.

5. Evaluations

The evaluation benchmarks and metrics are described in Sections 5.1 and 5.2 respectively. Each individual experiment is then detailed in its own subsection in Section 5.3. This is followed by a broader discussion in Section 6, including overall performance, run-times and future issues that need addressing.

All programs were implemented in Java on a 32-bit Ubuntu virtual machine with 3385 MB of RAM and a quad-core 2.40 GHz Intel 4700MQ i7 processor. The Student’s t-test\(^{21}\) was used for statistical significance testing at an \(\alpha\) value of 0.01.

5.1. Benchmarks

A summary of the test cases is provided in Table 1. Together, the test cases span over almost twenty domains, with six of the ten test cases being multi-class. Each test case contains two files, the goal being to find matching entity pairs in the files. Many of the cases have already been made publicly available by Semantic Web initiatives such as the instance matching track. The reference alignment for the experiments in this article was constructed at the University of Massachusetts. Both datasets are real-world and known to contain noise, although Rexa is believed to contain less noise than Eprints [72]. This dataset is also the most heterogeneous dataset in terms of properties, since Eprints contains far fewer properties (and also instances) than Rexa.

5.1.1. Test Cases 1, 2 and 3

The first three test cases, Persons 1, Persons 2 and Restaurants were first released by OAEI in 2010 and are described on the website\(^{24}\) as ‘real data cases’. In the literature, there is some source of confusion about this, with at least one paper describing them as synthetic [54]. Restaurants was originally a tabular dataset and is still widely used to evaluate record linkage systems\(^{25}\) [29]. Along with describing restaurants and people (for the Persons test cases), these datasets also contain instances from an Address class.

5.1.2. Test Case 4

Eprints-Rexa is another publicly available benchmark in the Semantic Web community [72]. Eprints\(^{26}\) is a small dataset containing information about papers produced within the AKT research project, while Rexa was extracted by the Rexa search server\(^{27}\) constructed at the University of Massachusets. Both datasets are real-world and known to contain noise, although Rexa is believed to contain less noise than Eprints [72]. This dataset is also the most heterogeneous dataset in terms of properties, since Eprints contains far fewer properties (and also instances) than Rexa.

5.1.3. Test Case 5

Test case 5, IM-Similarity, describes books and was generated from real-world data using crowdsourcing\(^{28}\). It was released quite recently (OAEI 2014), and the actual ground-truth has not been made available. To counter this, we manually created a reference alignment for the experiments in this article by using ad-hoc rules. Although the ad-hoc rules were framed to infer owl:sameAs links as closely as possible, there is always a possibility that the reference alignment contains noise. It is

---

\(^{21}\)Specifically, the paired t-test for sample means.


\(^{23}\)https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/unsupervised-im


\(^{25}\)In the record linkage literature, Restaurants is unambiguously considered real-world and not artificial [11].

\(^{26}\)eprints.aktors.org

\(^{27}\)www.rexa.info

Table 1: Details of benchmarks used in the evaluations. Each benchmark is a pair of files. The notation, where applicable, is (first dataset)/× (second dataset). Note that subject is counted among the Properties. Test cases 6 and 7 were artificially generated by SWING from underlying real-world IIMB data [71]

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Properties</th>
<th>Property alignments</th>
<th>Triples</th>
<th>Entity pairs</th>
<th>Matching entities</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Persons 1</td>
<td>15/14</td>
<td>15</td>
<td>9000/7000</td>
<td>2000 × 1000 = 2 million</td>
<td>500</td>
<td>2/2</td>
</tr>
<tr>
<td>2</td>
<td>Persons 2</td>
<td>15/14</td>
<td>15</td>
<td>10,800/5600</td>
<td>2400 × 800 ≈ 1.92 million</td>
<td>400</td>
<td>2/2</td>
</tr>
<tr>
<td>3</td>
<td>Restaurants</td>
<td>8/8</td>
<td>7</td>
<td>1130/7520</td>
<td>339 × 2256 = 764,784</td>
<td>89</td>
<td>2/2</td>
</tr>
<tr>
<td>4</td>
<td>Eprints-Rexa</td>
<td>24/115</td>
<td>24</td>
<td>4121/99,260</td>
<td>1130 × 18,492 ≈ 20.9 million</td>
<td>171</td>
<td>3/3</td>
</tr>
<tr>
<td>5</td>
<td>IM-Similarity</td>
<td>9/9</td>
<td>9</td>
<td>2204/2184</td>
<td>181 × 180 = 32,580</td>
<td>496</td>
<td>1/1</td>
</tr>
<tr>
<td>6</td>
<td>IIMB-059</td>
<td>31/25</td>
<td>23</td>
<td>9995/8979</td>
<td>1549 × 519 = 803,931</td>
<td>412</td>
<td>5/5</td>
</tr>
<tr>
<td>7</td>
<td>IIMB-062</td>
<td>31/34</td>
<td>30</td>
<td>9995/22,058</td>
<td>1549 × 265 = 410,485</td>
<td>264</td>
<td>5/5</td>
</tr>
<tr>
<td>8</td>
<td>Libraries</td>
<td>4/10</td>
<td>9</td>
<td>70,544/265,830</td>
<td>17,636 × 26,583 = 469 million</td>
<td>16,789</td>
<td>1/1</td>
</tr>
<tr>
<td>9</td>
<td>Parks</td>
<td>3/10</td>
<td>8</td>
<td>1701/3590</td>
<td>567 × 359 = 203,553</td>
<td>322</td>
<td>1/1</td>
</tr>
<tr>
<td>10</td>
<td>Video Game</td>
<td>11/4</td>
<td>4</td>
<td>220,000/48,132</td>
<td>20,000 × 16,755 = 335.1 million</td>
<td>10,000</td>
<td>1/1</td>
</tr>
</tbody>
</table>

thus more appropriate to interpret this task as a link discovery task rather than the more specific instance matching task. Note also that this test case contains multilingual property values.

5.1.4. Test Cases 6 and 7

Test cases 6 and 7 are over the film domain and were artificially generated from real movie data using SWING, which injects controlled degrees of heterogeneity into an underlying corpus of real-world IIMB movie instances [71]. The types of heterogeneity (value, structural and semantic) were described in a companion paper [73], and the datasets were introduced as instance matching OAEI benchmarks in 2010 (along with Persons and Restaurants). Eighty target datasets were generated by SWING from a common source. These were partitioned into four equal-sized folders, based on whether they contained only one of the three heterogeneities above, or all three (denoted as comprehensive heterogeneity in the OAEI report). We randomly picked two pre-generated SWING configurations (folder numbers 59 and 62 in the publicly available files) for the evaluations, with one containing only semantic heterogeneity (IIMB-059) and the other containing comprehensive heterogeneity (IIMB-062). Given the schema-free assumption, IIMB-059 is an interesting test of system performance when faced purely with semantic heterogeneity.

5.1.5. Test Cases 8, 9 and 10

Test case 8 describes US libraries. The first file was from a Point of Interest (POI) website that allows users to upload GPS data, and the second file was taken from a US government listing of libraries. Both files were extracted in the CSV format and were converted to RDF by treating each column name in the CSV file as a property.

Test case 9 is similar to test case 8 except it describes national parks in the United States. Although Libraries is much larger than Parks, both datasets exhibit similar challenges of schema heterogeneity, since the first file in both cases contains fewer properties than the second file. Another challenge is that, since both cases have files from POI websites, they contain longitude and latitude information. For many of the matching entity pairs, the values are not identical, which makes the task challenging for domain-agnostic instance matchers (such as the proposed system) that are not specifically configured for addressing the challenges of geo-locational data.

Finally, test case 10 describes video game information. The first file contains a sampling of video games extracted from DBpedia, while the second file was extracted as structured data (and converted to RDF triples in a manner similar to Libraries and Parks) from a reputable charting website. Similar to Libraries and Parks, it only contains instances from a single class. Additionally, a version of test case 10 was used recently in an orthogonal schema matching work [74].

5.2. Metrics

For all steps other than blocking (described subsequently), precision and recall were chosen as the metrics. For each of the concerned systems, assume a ground-truth set of true positives or TP. An algorithm would return its results as a set, which can be partitioned into the set of returned true positives, TP_R, and returned false positives, FP_R. Note that TP_R ≤ TP and the set difference TP – FP_R = TP. Precision and recall are defined as:

\[
\text{Precision} = \frac{|TP_R|}{|TP_R + FP_R|} \\
\text{Recall} = \frac{|TP_R|}{|TP|}
\]

The tradeoff between precision and recall can be expressed through their harmonic mean, denoted as the f-score.

The efficiency and effectiveness of blocking are evaluated by the special metrics Reduction Ratio and Pairs Completeness respectively [11]. Recall that the primary goal of blocking is to

---

29http://www.poi-factory.com/points
30Global Positioning System.
31Comma Separated Values.
32dbpedia.org
33vgchartz.com
34The harmonic mean of a and b is \(2ab/(a + b)\).
generate a candidate set of pairs that is small but has adequate coverage of the unknown ground-truth set of matching entity pairs. Reduction Ratio (RR) quantifies the criterion of efficiency and is given by the formula:

$$RR = 1 - \frac{|\Gamma|}{|\Omega|}$$  \hspace{1cm} (8)

Here, $\Omega$ is the full set of entity pairs (the Entity pairs column in Table 1), while $\Gamma$ is the candidate set of pairs generated by blocking. In the same vein, let $\Omega_m$ denote the ground-truth set of matching entities (the Matching entities column in Table 1). Pairs Completeness (PC) expresses effectiveness and is given by the formula below:

$$PC = \frac{|\Gamma \cap \Omega_m|}{|\Omega_m|}$$  \hspace{1cm} (9)

An efficiency-effectiveness tradeoff is commonly observed in real-world test cases. We use the f-score of RR and PC to express this tradeoff.

5.3. Experiments and Results

5.3.1. Experiment 1

**Goal:** The goal of this experiment is to evaluate the training set generator (TSG) proposed in Section 4.2.

**Setup:** To the best of our knowledge, the Dumas TSG is the only other current system that automatically detects heuristic duplicates in structurally heterogeneous datasets [42], and is thus used as the baseline in this experiment. Dumas uses logTFIDF to locate the desired set of duplicates, essentially comprising lines 1-5 of Algorithm 1. First, the precision-recall tradeoff offered by the Dumas TSG is plotted against that of the re-sorted list output by lines 6-7 of Algorithm 1 by using the Token-Jaccard score. We measure statistical significance by comparing the f-score series generated by the two systems using the paired t-test for sample means.

The curves are plotted by considering a range of values for the parameter $n$ in Algorithm 1. Note that later algorithms depend on $n$ since $n$ is used to tune the quality-representativeness tradeoff described in Section 4.2. Ideally, $n$ should be large enough to adequately represent the characteristics of the underlying dataset, but not so large that too many incorrectly labeled pairs get included in the generated training set. Towards this end, $n$ was chosen to equal 500 for the second part of the experiment. That is, the top 500 elements from the re-sorted list are picked as duplicates, such that no instance is repeated more than once\(^{35}\) (line 8 of Algorithm 1). The chosen duplicates are permuted (line 9 of Algorithm 1) to yield 500 non-duplicates. For fairness, the same procedure is conducted on the Dumas list and the resulting precision, recall and f-score of the 500 duplicates are tabularly reported. For the Dumas list, the results are reported both with and without the uniqueness constraint. Note that, because the data is both deterministic and single-valued (i.e. not obtained as a series unlike the previous experiment), statistical significance testing does not apply to this part of the experiment.

We set the parameter $\text{thresh}$ in Algorithm 1 to 0.01 (and in self-tuning mode; see Section 4.2) for all experiments. The self-tuning functionality was never invoked, indicating that the default value of $\text{thresh}$ is typically adequate, even across the wide variety of test cases.

Finally, the precision of the 500 non-duplicates generated through permutation is also reported. Given that the vast majority of entity pairs are non-duplicates, computing the recall of the generated non-duplicates training set serves no purpose, since it is expected to nearly equal 0. The permutations are conducted across ten independent trials for each test case, and averages and standard deviations (of the resulting non-duplicates precision) are both recorded.

**Results:** Figures 6 and 7 show the results of the proposed TSG against the Dumas TSG based on whether the highest f-score achieved by either method was above 60%. Except on Eprints-Rexa and Parks, the proposed TSG outperforms the Dumas TSG. Except on Parks, the performance difference between the systems is statistically significant, with the test conducted over the two f-score series.

Closer investigation of the anomalous Eprints-Rexa result showed that the problem arose because of schema mismatch. Rexa has 115 distinct property labels (counting both object and datatime properties; see Section 5.1), while Eprints only has 24 distinct property labels. While the logTFIDF distance measure was able to somewhat compensate for this mismatch, the subsequent Token-Jaccard measure that was used to re-sort the list in lines 6-7 of Algorithm 1 led to a decline in the overall results. To test the hypothesis that schema mismatch caused Token-Jaccard to perform so poorly, an additional experiment was conducted where the top 500 duplicates output by the initial run of the TSG on Eprints-Rexa was input to the hybrid property aligner (Algorithm 2). The properties that were absent in the alignment set output by the aligner were discarded and the TSG was re-run. The second figure in Figure 7 shows that this simple unsupervised step can be used to boost results in cases where the schema mismatch is large, even though this is not the primary purpose of the property aligner. Section 5.3.2 investigates the property alignment step in more detail.

Table 2 shows the precision, recall and f-score of the top 500 duplicates retrieved from the lists returned by the Dumas and proposed TSGs, such that no instance is ever repeated more than once in the set of duplicates. Since the original Dumas TSG does not apply the uniqueness constraint, the results of retrieving the actual top 500 duplicates (regardless of whether instances are repeated) from the Dumas list are also reported alongside. Note that the recall metric is computed differently in Table 2. Specifically, the number of true positives in the retrieved 500 duplicates is divided by the quantity $\min(500, |\Omega_m|)$, where $|\Omega_m|$ is the actual number of matching entities (Table 1), instead of $|\Omega_m|$ (as with traditional recall computation\(^{36}\)). The table shows that the proposed TSG equals or outperforms the

\(^{35}\)We denote this constraint as the uniqueness constraint.

\(^{36}\)The reason for bounding the denominator in this particular experiment is to prevent the recall from exceeding 100%.
Figure 6: Duplicates precision-recall results of the proposed TSG against the Dumas TSG for cases where the (overall) highest achieved f-score was above 60%.

Figure 7: Duplicates precision-recall results of the proposed TSG against the Dumas TSG for cases where the highest (overall) achieved f-score was below 60%. The second figure Eprints-Rexa (after re-running TSG) is explained in the text.

Table 2: Comparative results of the proposed TSG against Dumas, taking the top 500 duplicates with the uniqueness constraint. Also shown are the top 500 duplicates results for the original Dumas TSG (without the uniqueness constraint).

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Proposed TSG</th>
<th>Dumas TSG (constrained)</th>
<th>Dumas TSG (original)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall %</td>
<td>Precision</td>
<td>F-Score</td>
</tr>
<tr>
<td>1 Persons 1</td>
<td>75.20%</td>
<td>75.20%</td>
<td><strong>75.20%</strong></td>
</tr>
<tr>
<td>2 Persons 2</td>
<td>23.75%</td>
<td>19.00%</td>
<td>21.11%</td>
</tr>
<tr>
<td>3 Restaurants</td>
<td>100.00%</td>
<td>36.18%</td>
<td><strong>53.13%</strong></td>
</tr>
<tr>
<td>4 Eprints-rexa</td>
<td>43.40%</td>
<td>43.40%</td>
<td>43.40%</td>
</tr>
<tr>
<td>5 IM-Similarity</td>
<td>96.49%</td>
<td>92.18%</td>
<td><strong>94.29%</strong></td>
</tr>
<tr>
<td>6 IIMB-059</td>
<td>84.95%</td>
<td>80.83%</td>
<td><strong>82.84%</strong></td>
</tr>
<tr>
<td>7 IIMB-062</td>
<td>67.80%</td>
<td>67.80%</td>
<td>67.80%</td>
</tr>
<tr>
<td>8 Libraries</td>
<td>100.00%</td>
<td>100.00%</td>
<td><strong>100.00%</strong></td>
</tr>
<tr>
<td>9 Parks</td>
<td>73.60%</td>
<td>66.95%</td>
<td>70.12%</td>
</tr>
<tr>
<td>10 Video Game</td>
<td>88.60%</td>
<td>88.60%</td>
<td><strong>88.60%</strong></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>75.38%</strong></td>
<td><strong>67.01%</strong></td>
<td><strong>69.65%</strong></td>
</tr>
</tbody>
</table>

Other systems on six datasets. On two of the remaining datasets, its f-score is within 6% of the winning f-score and on average, the proposed system outperforms the baselines on all metrics.

Finally, the set of 500 non-duplicates generated by permuting the set of duplicates obtained from the three systems in Table 2 had high overall quality, with average precision on all test cases (and for all three systems) at least 98%, and with less than 1% standard deviation across ten independent trials per test case and system. At a p-value of less than 0.01, the difference between the three setups was not found to be statistically signifi-
5.3.2. Experiment 2

Goal: The goal of this experiment is to evaluate the property aligner in Section 4.3.

Setup: The 500 duplicates and non-duplicates output by the proposed TSG are input to the hybrid property aligner described by Algorithm 2. Two baselines are used in this experiment to illustrate the benefits of a hybrid aligner. The first baseline is the Dumas schema matcher [42], which uses the noisy duplicates generated by the Dumas TSG (without the uniqueness constraint). The matcher computes a similarity matrix for each of the $n$ duplicates, and then aggregates them into a single matrix on which the Hungarian algorithm is run [75]. The Hungarian algorithm is a generic procedure which assigns a different row to each column, such that the total sum of values in the chosen cells is maximized over all valid assignments. The Hungarian algorithm has cubic complexity in the number of columns, which makes it expensive for large numbers of properties. Because the Hungarian algorithm cannot assign the same row to two different columns, Dumas can only output an alignment set with global 1:1 cardinality. When evaluating Dumas, a full parameter sweep was conducted to ensure optimal performance. Thus, the number of generated duplicates was not fixed at 500 for Dumas, but tuned for each test case. Only the optimal results are reported for Dumas.

The second baseline is denoted as the Column Matcher, and uses similar principles as property aligners proposed in recent Semantic Web instance matchers (e.g., Raven [26]). The Column Matcher directly constructs a single similarity matrix by computing the Token-Jaccard score of all values in two columns corresponding to a cell of the similarity matrix. Unlike Dumas and the proposed aligner, the Column Matcher does not use a training set. Once the similarity matrix is constructed, all values above a threshold are output as a match. Similar to the evaluations over Dumas, a full sweep is conducted over the threshold range [0, 1] to ensure optimality. Note that the parameter sweeps confer an empirical advantage on both baselines, since the proposed aligner takes as arguments the top 500 samples (with the uniqueness constraint) output by Algorithm 1, and is parameter-free. Note that on Eprints-Rexa, the top 500 samples output by the original TSG run (not the re-run; see Figure 7) are passed as arguments to Algorithm 2 to avoid biasing the results.

Results: The results of property alignment are tabulated in Table 3. The superior performance of both Dumas and Algorithm 2 against the Column Matcher presents a strong case for the use of instance information (even with noise present) when aligning the properties. Note that, although Algorithm 2 is outperformed by the baselines on six of the test cases, it is more balanced in its precision-recall tradeoff and outperforms, on average, both baselines by over 10% in terms of f-score. The proposed aligner scores below 75% on precision on only two of the datasets (Libraries and Parks), and never below 75% on recall. Dumas scores below 75% on recall on half the datasets. The Column Matcher is even more skewed, with less than 75% recall on eight of the datasets. Given that the alignment set is not intended to be used as an output in itself but for building a tractable feature space (Section 4.4), we believe that this distinction is important, since property alignment recall is more important in the context of the overall instance matching task than precision. Intuitively, recall matters more than precision because not every feature in the feature space (utilized by subsequent learning algorithms) has to be high-performing given that feature-selection is executed on the constructed space (Algorithm 4), but the absence of good features (due to low recall) potentially leads to classifiers and blocking schemes being less discriminative. A last point to note is the proposed aligner’s robustness to noise, as exhibited by the performance on Eprints-Rexa, where the generated set of 500 duplicates had an f-score below 50% (Table 2).

5.3.3. Experiment 3

Goal: The goal of this experiment is to evaluate the DNF blocking scheme learner (DNF-BSL) in Section 4.5.1.

Setup: As a preliminary experiment, we ascertain if advanced blocking techniques are even warranted in real-world cases, or if a simple token-based clustering approach suffices, by running the classic Canopies algorithm on each of the ten test cases [76]. Canopies uses a threshold and, an inexpensive distance metric (typically TFIDF [58]). The algorithm works by using each entity from the first file as a seed entity to represent a cluster. All entities from the second file with distance (using the inexpensive distance metric less than the threshold parameter) are assigned to that cluster. Since an entity in the second file can be paired with multiple seed entities from the first file, the clusters overlap like canopies (giving the algorithm its name). In the Relational Database (RDB) community, Canopies was found to deliver excellent performance on many test cases [77].

In the main experiment, Algorithm 4 is evaluated against the trigrams-based Attribute Clustering (AC) baseline [13]. AC is considered to be a state-of-the-art unsupervised blocking approach for schema-free data represented only as a set of attribute-value pairs. The method extracts trigrams from each attribute value in the dataset, and then clusters attributes by computing the overlap between their trigram value-sets.

Recall that the DNF blocking scheme learner in Algorithm 4 required the setting of two parameters $k$ and $\kappa$. Since the algorithm is exponential in $k$ and previous results have not found large differences between the $k = 1$ and $k = 2$ settings [67], $k$ is set to 1 in the remainder of this article. Similar to the parameter $\text{thresh}$ in Experiment 5.3.1, $\kappa$ was set to a default value of 0.2 and in self-tuning mode with a decrement of 0.05. In the majority of the cases, the self-tuning mode was not invoked. In

---

37 Without loss of generality, assume that the number of columns is no greater than the number of rows.

38 That is, each property can participate in at most one match.

39 Even on the precision metric, both baselines score below 75% roughly half of the time, exhibiting the unpredictable nature of their performance.

40 Technically, it uses two thresholds, but assigning them a common value was found to yield the best empirical results [77].
Table 3: Comparative results of Algorithm 2 against the Column Matcher and Dumas baselines

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Proposed Aligner</th>
<th>Dumas</th>
<th>Column Matcher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall</td>
<td>Precision</td>
<td>F-Score</td>
</tr>
<tr>
<td>1 Persons 1</td>
<td>80.00%</td>
<td>100.00%</td>
<td>88.89%</td>
</tr>
<tr>
<td>2 Persons 2</td>
<td>85.71%</td>
<td>80.00%</td>
<td>82.76%</td>
</tr>
<tr>
<td>3 Restaurants</td>
<td>85.71%</td>
<td>100.00%</td>
<td>92.31%</td>
</tr>
<tr>
<td>4 Eprints-Rexa</td>
<td>100.00%</td>
<td>92.31%</td>
<td>96.00%</td>
</tr>
<tr>
<td>5 IM-Similarity</td>
<td>100.00%</td>
<td>81.82%</td>
<td>90.00%</td>
</tr>
<tr>
<td>6 IIM-059</td>
<td>100.00%</td>
<td>82.14%</td>
<td>90.19%</td>
</tr>
<tr>
<td>7 IIMB-062</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>8 Libraries</td>
<td>100.00%</td>
<td>22.50%</td>
<td>36.73%</td>
</tr>
<tr>
<td>9 Parks</td>
<td>100.00%</td>
<td>26.67%</td>
<td>42.11%</td>
</tr>
<tr>
<td>10 Video Game</td>
<td>75.00%</td>
<td>75.00%</td>
<td>75.00%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>92.60%</strong></td>
<td><strong>76.04%</strong></td>
<td><strong>79.40%</strong></td>
</tr>
</tbody>
</table>

Table 4: Blocking results on the ten datasets using Canopies [76]. RR and PC were defined in Section 5.2

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>PC</th>
<th>RR</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Persons 1</td>
<td>100.00%</td>
<td>97.96%</td>
<td>98.97%</td>
</tr>
<tr>
<td>2</td>
<td>Persons 2</td>
<td>99.75%</td>
<td>98.58%</td>
<td>99.16%</td>
</tr>
<tr>
<td>3</td>
<td>Restaurants</td>
<td>75.28%</td>
<td>99.39%</td>
<td>85.67%</td>
</tr>
<tr>
<td>4</td>
<td>Eprints-Rexa</td>
<td>6.32%</td>
<td>99.99%</td>
<td>11.89%</td>
</tr>
<tr>
<td>5</td>
<td>IM-Similarity</td>
<td>26.90%</td>
<td>89.99%</td>
<td>41.42%</td>
</tr>
<tr>
<td>6</td>
<td>IIM-059</td>
<td>100.00%</td>
<td>96.98%</td>
<td>98.47%</td>
</tr>
<tr>
<td>7</td>
<td>IIMB-062</td>
<td>51.89%</td>
<td>98.09%</td>
<td>67.87%</td>
</tr>
<tr>
<td>8</td>
<td>Libraries</td>
<td>87.74%</td>
<td>99.99%</td>
<td>93.47%</td>
</tr>
<tr>
<td>9</td>
<td>Parks</td>
<td>0%</td>
<td>100.00%</td>
<td>0%</td>
</tr>
<tr>
<td>10</td>
<td>Video Game</td>
<td>69.24%</td>
<td>99.92%</td>
<td>81.80%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>61.71%</strong></td>
<td><strong>98.09%</strong></td>
<td><strong>67.87%</strong></td>
<td></td>
</tr>
</tbody>
</table>

two cases (Eprints-Rexa and Libraries), the self-tuning was invoked and the value of $\kappa$ at which the algorithm succeeded was 0.01. Either way, the algorithm was able to successfully output a DNF blocking scheme. We evaluate the blocking results using the PC, RR and f-score metrics that were introduced in Section 5.2.

Results: Table 4 shows the results of the preliminary experiment. While Canopies achieves over 90% f-score on four test cases, its general performance exhibits much deviation. On Parks, the method fails completely, achieving 0% PC. Additionally, the algorithm was found to run quite slowly on large datasets, and the threshold parameter had to be tuned separately for each run. These results show that the performance of Canopies is unpredictable for schema-free RDF data. Its average achieved f-score (67.87%) is also quite low compared to the state-of-the-art, as the following experiment will demonstrate.

Table 5 shows the results for the main experiment, where the proposed DNF-BSL is evaluated against the Attribute Clustering baseline. Both methods perform quite well generally, although the proposed system outperforms the baseline on six of the test cases, and by 1.5% f-score on average. An important difference between the systems’ performances is that the proposed method tends to favor the RR metric over PC. While the f-score treats PC and RR equally, blocking practitioners have argued that even small differences in RR can be consequential [11]. This is because (see Equation 8) RR is measured over the full set of entity pairs, which is a quadratic function and can number in the millions even for moderately sized datasets (evidenced by the Entity pairs column in Table 1). In contrast, PC is a linear function of the matching entity pairs in the files, which are typically quite small in number. By this argument, low values of RR can lead to the overall instance matching task becoming intractable in a practical implementation. Table 5 shows that the proposed DNF-BSL only achieves RR below 95% on two datasets (and never below 90%), while the baseline can be slightly more unpredictable (less than 95% RR on four datasets).

5.3.4. Experiment 4

Goal: This experiment evaluates three Support Vector Machines (SVMs)\(^{42}\) against each other, with the goal of determining how the degree of supervision (the number of samples an SVM is trained on) and noise (incorrect labeling of training samples) affect overall classification performance. We also evaluate the SVMs against an unsupervised baseline method that combines Locality Sensitive Hashing (LSH) and Expectation Maximization (EM) [39, 41], which have been successfully applied to both instance and ontology matching in the recent past [40, 78].

Setup: As a first step, the candidate set of pairs for the classification step is generated using the blocking approach that had the higher f-score in the previous experiment. An SVM is trained using the 500 duplicates and non-duplicates generated

\(^{41}\)100% PC and 0% RR can also be achieved if the threshold parameter is set high enough. Even with exhaustive parameter sweeps, no other value sets were obtained except for these two extremes.

\(^{42}\)All SVMs in this paper use RBF (Radial Basis Function) kernels, for which an efficient implementation may be found in the LibSVM library [69]; see Section 4.5.2.
Table 5: Comparative results of proposed DNF blocking scheme learner (DNF-BSL) in Algorithm 4 and the Attribute Clustering baseline. RR and PC were defined earlier in Section 5.2

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Proposed DNF-BSL</th>
<th>Attribute Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC</td>
<td>RR</td>
</tr>
<tr>
<td>1  Persons 1</td>
<td>100.00%</td>
<td>99.75%</td>
</tr>
<tr>
<td>2  Persons 2</td>
<td>99.00%</td>
<td>99.79%</td>
</tr>
<tr>
<td>3  Restaurants</td>
<td>100.00%</td>
<td>99.73%</td>
</tr>
<tr>
<td>4  Eprints-Rexa</td>
<td>98.16%</td>
<td>99.28%</td>
</tr>
<tr>
<td>5  IM-Similarity</td>
<td>100.00%</td>
<td>98.14%</td>
</tr>
<tr>
<td>6  IIMB-059</td>
<td>99.76%</td>
<td>93.35%</td>
</tr>
<tr>
<td>7  IIMB-062</td>
<td>47.73%</td>
<td>98.11%</td>
</tr>
<tr>
<td>8  Libraries</td>
<td>97.96%</td>
<td>99.99%</td>
</tr>
<tr>
<td>9  Parks</td>
<td>95.96%</td>
<td>94.41%</td>
</tr>
<tr>
<td>10 Video Game</td>
<td>98.73%</td>
<td>99.96%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>93.73%</td>
<td>98.25%</td>
</tr>
</tbody>
</table>

by the proposed TSG. Let this SVM be denoted as Unsupervised, since the training sets are automatically generated. Two supervised SVMs are trained on 10% and 50% of the ground-truth. These perfectly labeled samples are used for both training and cross-validation, with the rest of the ground-truth not seen by the classifier till actual testing time to avoid bias. We compare SVM performances using precision-recall graphs.

The supervised SVMs are expected to illustrate the limits of the unsupervised system by showing how the numbers (and levels of noise) in the samples affect the training of the SVMs in the described feature space. Section 4.5.2 provided a rationale for why SVMs were expected to be advantageous for the problem context compared to other adaptive classifiers.

The alternate unsupervised baseline is set up as an Expectation Maximization (EM) clustering procedure as follows. First, to improve robustness and reduce processing times, we reduce the original feature space by computing, for each feature vector in the candidate set, seven hashes\(^43\) using an open-source Locality Sensitive Hashing (LSH) package\(^44\). Thus, a new feature-vectors file was produced, where each vector is represented by seven real numbers. Appropriately formatted, this file was then input to the EM algorithm implemented in the Weka\(^45\) machine learning package. To maximize performance, we specified that all vectors had to be (probabilistically) clustered into two clusters. The final step was to map the two clusters to classes (that is, duplicates or non-duplicates). This was achieved by using a simple but effective heuristic: the larger cluster was considered to map to the non-duplicates class. Note that this mapping was found to maximize this baseline’s metrics. Finally, note that, as the alternate baseline is independent of the noisy training set, it provides a good test of how well simple heuristics or traditional techniques (such as EM) fare on schema-free instance matching.

Results: Figure 8 shows the results for the cases where the highest achieved f-score (for any of the systems) was greater than 60%. On four test cases (Libraries, Restaurants and both Persons cases), there is no statistically significant difference between either of the supervised systems. On all these datasets, the SVM is able to adapt to the unseen data without much supervision. The IIMB-059 illustrates that this is not necessarily the case for every test case. The overall results also show that the SVM is able to adapt even when instances from multiple classes are present. In Figure 8, only the Libraries and IM-Similarity test cases have instances from a single class.

Although the unsupervised SVM does not generally perform as well as the supervised SVMs, it is still competitive on three of the test cases (Persons 1, Restaurants and Libraries) over a particular range of precision and recall. On IIMB-059, Unsupervised outperforms Supervised 10% in terms of the highest f-score. We explore an iterative approach in the next experiment to improve the unsupervised SVM performance on some of these test cases.

Figure 9 illustrates the four test cases for which no SVM manages to achieve a high f-score. In all cases, the unsupervised SVM performs at least as well as (and on Video Game and Eprints-Rexa, outperforms) one of the supervised SVMs. A closer look at the results showed that all the SVMs were returning many false positives for these cases. We hypothesize that the SVMs were overfitting the data on these four test cases.

To study this hypothesis, it is instructive to compare the graphs in Figures 8 and 9 with the results of training set generation (TSG) in Experiment 5.3.1. On some cases, particularly Parks, Video Game and even Eprints-rex, the TSG outperforms even the supervised SVMs by a considerable margin. This is most apparent in the Video Game case, where the Supervised 50% SVM performs the worst. Automatically determining when to choose the TSG over the adaptive classifier is an important issue for future work, and directly related to the recent research interest in self-configuring systems [79]. This issue is discussed further in Section 6. We note that at least one other instance matching study also made a similar finding:

\(^{43}\)CosineHash, three variants of EuclideanHash, and three variants of CityBlockHash.
\(^{44}\)https://github.com/JorenSix/TarsosLSH
\(^{45}\)http://www.cs.waikato.ac.nz/ml/weka/
Figure 8: Precision-recall results of the three SVMs and an alternate unsupervised baseline for cases where the highest (overall) achieved f-score was greater than 60%. On Libraries, Restaurants and both Persons test cases, the two supervised systems performed near-identically, and the difference is not statistically significant. On Persons 2 and IIMB-059, the highest f-score achieved by the alternate baseline was near-0%.

Figure 9: Precision-recall results of the three SVMs and an alternate unsupervised baseline for cases where the highest (overall) achieved f-score was below 60%. On Eprints-Rex and Parks, Unsupervised and Supervised (10%) are near-coincidental. On IIMB-062, Supervised (10%) exhibits near-0% f-score throughout. Note that, except on IIMB-062, the alternate baseline exhibits near-0% f-score throughout.

namely, that an adaptive classifier is not a silver bullet for every instance matching test case [27]. Finally, the alternate unsupervised baseline is outperformed by all methods on the majority of the cases. There are only two exceptions: low recall levels in the IM-Similarity test case (where it is briefly competitive with all methods), and the IIMB-062 test case, where it outperforms the Supervised (10%) SVM. The results show, quite unambiguously, that simple heuristics-based, distance-based and clustering-based techniques are not adequate by themselves, or even in simple combinations, for noisy schema-free RDF data. This may explain why the instance matching literature covering these methods makes strong assumptions about the underlying datasets, including existence of structure and meta-data (e.g. ontologies) [40, 78, 41]. Adapting these traditional methods so that they perform well on schema-free RDF datasets is left for future work.

5.3.5. Experiment 5

Goal: This experiment explores an iterative approach for improving the performance of the unsupervised SVM and the alternate approach in the previous experiment.

Setup: The previous experiment showed that, in at least five test cases (see the graphs for Persons 1, Persons 2, Restaurants, IM-Similarity and Libraries in Figure 8), the SVM trained on 10% of the ground-truth was able to achieve better performance than the unsupervised SVM, despite being trained on fewer samples (but without noise). It would thus seem that in these cases, the noise (more than the size of the training set) dictates SVM performance. In this experiment, we explore this hypothesis and show that the effects of this noise can be accounted for, while still keeping the system unsupervised, if an iterative approach is adopted. Namely, the SVM re-trains itself.
on a small set of top-scoring duplicates initially output by it, after which the classification step is re-run\textsuperscript{47}. Specifically, the 50 most confident samples output by the unsupervised SVM are first permuted to obtain 50 non-duplicates (line 9 of Algorithm 1), which are together used to re-train the SVM. We choose a much smaller number than that\textsuperscript{48} used in the first pass to skew the quality-representation tradeoff in favor of quality. The expectation is that, in the cases where Supervised 10\% outperformed Unsupervised in the previous experiment, the gap between the two systems will significantly narrow, if not eliminated altogether. On the datasets where representation mattered more than quality, the performance is expected to decline. One example of the latter case is IIMB-062 (Figure 9).

In order to test the post-iteration performance of the alternate baseline, we used the ClassificationViaClustering facility (available as a class in the Java implementation) in the latest version of Weka. This facility allows a practitioner to use labeled instances to inform the clustering of unseen data. Since only 100 labeled samples (50 duplicates and 50 non-duplicates\textsuperscript{49}) are being used, we do not expect the clustering to be radically different. Other details on how the clustering was conducted and instances were classified can be found in the Setup sub-section of the previous experiment.

Results: Figure 10 compares the two unsupervised runs (before and after iteration). In six of the ten test cases, the highest achieved f-score improved after iteration. After iteration, the performance difference between the unsupervised system and both supervised systems (from the previous experiment) on Persons 1 narrows so that there is no statistically significant difference between the three systems (the post-iteration SVM and the supervised SVMs from the previous experiment). Near-perfect results are observed, showing that (on Persons 1) training set noise had a much greater impact on SVM performance than training set size.

The opposite is true for both the IIMB datasets, and more surprisingly, Persons 2. The results on Persons 2 were surprising because, as stated earlier, Supervised 10\% achieved excellent performance on it. In further experiments, we re-trained the SVM on larger sample numbers (ranging from 10–400), to test if the number of samples predominately affects performance. Even with this tuning, the post-iteration SVM never outperformed the pre-iteration SVM in our experiments. We believe that the low precision of the pre-iteration SVM on Persons 2 is the primary cause behind the post-iteration SVM’s performance decline. The SVM is not able to compensate for the high levels of noise even in the most confident samples retrieved by the pre-iteration SVM on Persons 2, regardless of training set size. This shows that the iterative procedure is not always sufficient; in particular, it can lead to a decline in overall performance if the original (first-pass) SVM outputs extremely low-quality results to begin with.

\textsuperscript{47}That is, the same candidate set from the previous experiment (for each of the test cases) is re-classified.

\textsuperscript{48}This number was 500, as described in Experiment 1.

\textsuperscript{49}There is a small caveat: for the alternate baseline, the 50 duplicates were not permuted to yield 50 non-duplicates. Instead, we took the 50 lowest-ranked samples from the probabilistically scored candidate set in Experiment 4.

On the IIMB datasets, we note that the post-iteration curves resemble those of Supervised 10\% from the previous experiment. On IIMB-062 in particular, the curve ‘collapses’, showing that the number of samples is the determining factor on performance. To test this claim, we re-trained the SVM on the top 200 samples instead of the top 50 samples. Figure 11 shows the results of this supplementary experiment, where the post-iteration SVM now outperforms the original SVM and also Supervised 50\% at low recall levels. Note that, for this specific experiment, the unsupervised alternate baseline was not evaluated.

We also note that on IM-Similarity and Parks, the improvements are quite drastic, with the post-iteration unsupervised system effectively outperforming both supervised systems from the previous run. On these test cases, the iteration achieves its maximum utility. On the other two cases Eprints-Rexa and Video Game, iteration also improves performance but by a near-indistinguishable margin.

Finally, we note that the iteration did not improve (or otherwise modify) the clustering procedure at all; hence, the post-iteration and pre-iteration performance of the unsupervised LSH are coincidental. This can be attributed to the relative stability of EM, especially given the very small labeled set that was provided to the system. Furthermore, the distinguishing characteristics (that is, different feature values) of individual instances in the training set are neutralized by the LSH feature-reducing computations and provide less information to the EM procedure than is provided to the SVMs. The net result is that there is no improvement in baseline performance. This finding also implies that the baseline may not be amenable to techniques such as active learning where a user is continuously trying to improve the system through incremental labeling.

6. Discussion

The previous section evaluated four unsupervised configurations of the system on ten test cases. These configurations are the proposed Training Set Generator (TSG), the Dumas TSG (TSG[D]), and the two unsupervised SVMs (both before and after iteration). We denote these as configurations because, from the perspective of a practitioner, these may be thought of as options that yield a set of duplicates as output. As we earlier
observed, it is not always the case that the full system (using the SVM) outperforms the TSGs. On certain test cases (such as Eprints-Rexa), the TSGs are higher-performing configurations because the SVMs tend to overfit the data. The experiments also showed results for two supervised configurations; namely, the results output by the two SVMs trained on 10% and 50% of the perfectly labeled ground-truth respectively.

To synthesize the experimental findings, we performed an analysis and compared the highest achieved f-score (by any of the six configurations) on all ten test cases. The analysis showed that on five of the ten test cases (Eprints-Rexa, IIMB-062, Libraries, Parks and Video Game), the best unsupervised configuration outperformed the best supervised configuration, while on one of the test cases (Persons 1), both configuration classes achieved roughly the same highest f-score (99.90%-100%). On the four test cases where the best supervised configuration outperformed the best unsupervised configuration on the highest achieved f-score metric, the best unsupervised configuration achieved a highest f-score that was within 5% of the highest f-score achieved by the best supervised configuration on two of the test cases (IIMB-059 and IM-Similarity). These results show that using an unsupervised configuration for schema-free RDF instance matching is a promising alternative compared to a manually intensive duplicate-labeling process, followed by the execution of a supervised configuration. An interesting issue is the choice of the configuration, which is clearly guided by the dataset characteristics. We describe this issue as a promising line of future research in Section 7.

In this vein, we conduct a finer-grained analysis of the SVM outputs of the system to better synthesize their dependence on the datasets. Specifically, we report the highest-achieved f-scores as well as corresponding precision and recall metrics for the best performing SVM configuration (both supervised and unsupervised) in Table 6.

Table 6 shows that, while the supervised SVMs (typically, SVM 50%) perform the best on average, the unsupervised...
SVMs are roughly at par in terms of recall (0.34% difference) and are respectively within 9% and 8% in terms of precision and F-score by comparison. Interestingly, in the three cases where an unsupervised SVM configuration is at par with (or outperforms) a supervised SVM configuration, the configuration is the post-iteration unsupervised SVM. On average, neither unsupervised configuration has an edge over the other.

We also note that, since both Persons test cases and Restaurants were used in the OAEI 2010 instance matching track for evaluations, it is possible to compare the results (on those benchmarks) of the proposed system against the reported highest F-scores in the competition. Several systems competed in the instance matching track of OAEI 2010, including ASMOV [48], CODI [50], RiMOM [49], ObjectCoref [30] and LN2R [80], some of which were described in Section 2. We note that none of these systems are schema-free, and all take the available meta-data into account when linking entities. The results show that the highest F-scores achieved in the competition on Persons 1, Persons 2 and Restaurants were 100%, 97% and 81% respectively, with the RiMOM ontology matcher achieving the highest scores on all three\(^5\). We note that the post-iteration SVM was able to achieve 99.90% and 93.68% highest F-score on Persons 1 and Restaurants respectively, while the TSG configuration of the system achieved 67.76% highest F-score on Persons 2. The results in Table 6 are also consistent with those obtained in a recent empirical work by Soru and Ngomo which showed that supervised machine learning classifiers are capable of simultaneously achieving F-scores above 90% on both Persons datasets and Restaurants [27].

We end with a note on the run-times of various system components. Except for the classification step\(^3\), the full system ran in less than three minutes for all the datasets. This includes the summed times taken for training set generation, property alignment and the training of the two learners (blocking scheme and SVM). Within those three minutes, the proportion of time taken by a component depended on the test case. For example, Eprints-Rexa took the most time on property alignment (more than 40%), while the learners proved to be the most expensive for the IIMB datasets. Since the number of training samples was fixed after training set generation, these discrepancies are best explained by the different numbers of properties (and property alignments) in each test suite. This provides a reason for why property alignment was most expensive for Eprints-Rexa.

The training set generation depended both on the numbers of properties as well as the number of entities in each file of the test case. Unsurprisingly, it was most expensive for the Libraries test case, but its run-time still did not exceed a minute.

The classification step was the most expensive step for all the test cases, which is quite typical for many instance-matchers [6]. While the run-time was near instantaneous for Persons 1 and 2, Restaurants, Parks and IM-Similarity, it was non-trivial for cases like Video Game, where it varied (across the experimental trials) from 20-30 minutes. Note that the classification step run-time never exceeded 30 minutes for any of the test cases. This is attributed to the high reduction ratios achieved by both the learned blocking scheme and the Attribute Clustering method on all test cases (Section 5.3.3). As a simple test of this claim, an SVM classifier was run on the full set of entity pairs on the IIMB-062 dataset (the smallest test case after Parks), in terms of entity pairs; see Table 1). The recorded run-time for the classification was over five hours. This demonstrates the necessity of high-quality blocking. The observations also demonstrate, from an empirical standpoint, that the unsupervised learning step (Step 0 in Figure 2) proposed in this article is subsumed by the two-step instance matching task, which includes only the blocking (or candidate set generation) and classification steps as its components.

### Table 6: Comparative Results of the Best Supervised SVM Configurations (SVM 10%, SVM 50%) against the Best Unsupervised SVM Configurations (SVM, SVM[II]), Where SVM Stands for the Original (First-Pass) Unsupervised SVM, and SVM[II] Stands for the Post-Iteration Unsupervised SVM Trained on the Top 50 Samples

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Unsupervised SVM Configuration</th>
<th>Supervised SVM Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Persons 1</td>
<td>100.00%</td>
<td>99.80%</td>
</tr>
<tr>
<td>Persons 2</td>
<td>87.00%</td>
<td>30.37%</td>
</tr>
<tr>
<td>Restaurants</td>
<td>100.00%</td>
<td>88.12%</td>
</tr>
<tr>
<td>Eprints-Rexa</td>
<td>17.52%</td>
<td>48.19%</td>
</tr>
<tr>
<td>IM-Similarity</td>
<td>83.04%</td>
<td>78.89%</td>
</tr>
<tr>
<td>IIMB-059</td>
<td>78.16%</td>
<td>63.89%</td>
</tr>
<tr>
<td>IIMB-062</td>
<td>41.29%</td>
<td>75.17%</td>
</tr>
<tr>
<td>Libraries</td>
<td>93.25%</td>
<td>61.84%</td>
</tr>
<tr>
<td>Parks</td>
<td>81.99%</td>
<td>55.11%</td>
</tr>
<tr>
<td>Video Game</td>
<td>33.84%</td>
<td>25.35%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>71.61%</td>
<td>62.67%</td>
</tr>
</tbody>
</table>

\(^5\)Available in the report published on the website of the 2010 Ontology Matching workshop, which hosts the OAEI competition: http://on2010. ontologymatching.org/

\(^3\)The performance of ObjectCoref has since improved on Restaurants, and was shown to outperform RiMOM's performance; the maximum f-score was still below 90%.

\(^3\)That is, the actual evaluation of the candidate set by a trained SVM.
7. Conclusion and Future Work

This article presented an instance matching system designed to locate pairs of matching entities in schema-free data. The system operates in three steps, with the first step being an automatic learning step that generates a heuristic training set and aligns properties on the basis of that set. The aligned properties and training set are used to generate features in a tractable space. A Disjunctive Normal Form Blocking Scheme Learner (DNF-BSL) is learned in that feature space, together with an RBF-kernel SVM classifier. The former is used in the blocking phase of instance matching, while the latter is used in the classification phase. Note that an advantage of separately generating a candidate set and training a classifier is that the system can be applied to online instance matching, where entities are constantly getting inserted, deleted or updated on the fly, such as on the Web of Linked Data. Since the system is a modular composition of relatively independent components, end-users have the ability to use the setup to build their own workflows. We provide this flexibility in the design of the system to encourage its adoption among Linked Data practitioners.

The experimental results show that taken together, the various unsupervised configurations can deliver generally competitive performance on many of the test cases. Individual components of the system, including the training set generator, the hybrid parameter-free property aligner and the DNF blocking scheme learner, are also shown to effectively compete against state-of-the-art alternatives. An additional advantage of some of the algorithms is that their parameters (thresh and $\kappa$ in Algorithms 1 and 2 respectively) are self-tuning and found to work well at default values. Finally, the experiments show that the system is able to gracefully handle multiple class instances without additional preprocessing, and the good results on IM-Similarity (see Figure 10) show that the system can also handle multilingual instances.

An important line of future work is to automatically determine when to use only the TSG output, and when to invoke the SVM, since the classification step can be expensive as candidate sets increase in size. This self-configuration is related to the effort to automatically tune the various parameters of the system to maximize expected performance. Such self-configuration is crucial to real-time, robust deployment on Linked Open Data.

References


[54] T. Joachims, Making large scale svm learning practical.


