Chapter 3
Entity Resolution

3.1 Introduction

Entity Resolution (ER) is the problem of devising *algorithmic* solutions for determining when two entities refer to the same underlying entity [66]. The problem is very common in almost all communities that deal with a lot of data, including knowledge discovery and data mining, databases and the Semantic Web. It is also a hard problem, despite (or perhaps because) of its *common-sense* nature, since it generally does not take specialized knowledge for a human being to answer the question of when two things are the same. ER problems widely exist in both industrial and non-industrial applications, and big technology companies often task entire teams to address the problem in its various guises. Multiple commercial and research solutions exist, some based on work that was originally done many decades ago [41]. Many books and special issues have also been dedicated to the topic. Although not humanly possible to cover ER in all its depth in this chapter, we attempt to synthesize the field in a conceptually meaningful way that will provide practical insights into why ER should be given special attention in any robust domain-specific KGC pipeline.

By way of a running example, consider the illustration in Fig. 3.1. Let us optimistically assume that complete and correct named entity recognition and relation extraction systems were applied to a corpus, yielding knowledge graph fragments. Clearly, the two nodes Nadal and Rafael Nadal extracted from the two documents need to be *resolved* since they are referring to the same underlying entity. In general, the problem is not unique to natural language sources, and can emerge even when we are constructing knowledge graphs over semi-structured (or even structured) raw sources like log files and XML. That being said, the natural language version of the problem is still special since one could potentially use linguistic clues to determine when extracted pronouns in a document refer to the same entity (anaphora or co-reference resolution). When extractions must be linked across documents, the problem is generally referred to as *cross-document coreference*

M. Kejriwal, *Domain-Specific Knowledge Graph Construction*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-3-030-12375-8_3

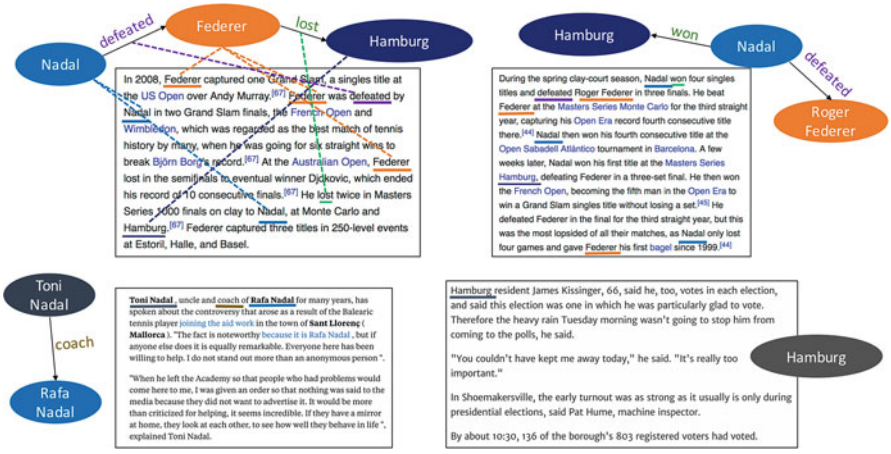

Fig. 3.1 An illustration of the cross-document ER problem. Only some extractions are shown from each of the documents, with the same color used if the nodes should be resolved together. The first document illustrates the 'provenance' of the KG nodes and relations. In general, each KG node or relation can trace its provenance to a *set* of *co-referenced* extracted mentions

resolution. Just because the document sources are separate doesn't preclude the use of linguistic cues and features.

However, the problem of ER is much broader, and the earliest known instances of it emerged in the patient linking and biomedical literature almost 50 years ago [123]. Even today, much more work has been done on the structured version of the problem (in both the database and Semantic Web communities) than in the natural language community [57, 96]. Much of the work in the structured data communities can be synthesized in a somewhat unified manner using similar concepts and terms. This synthesis, though brief, will be the focus of this chapter. Because of the nature of English language data as opposed to structured data, the NLP community has been forced to take a somewhat different approach to the problem. A good review of co-reference resolution may be found in [122].

In much work on ER, it was often the case that single-source, single-schema ER (often called *deduplication*) was the main focus of the research, along with close variants such as multi-source, single-schema ER. Recent research has attempted to address the multi-schema version of the problem [89], especially in KG-centric communities like the Semantic Web.

3.2 Challenges and Requirements

Before diving into solutions, we provide some intuition on why ER has proven so difficult to automate and algorithmically encode. Figure 3.2 provides some insight, despite its simplicity. The most important challenge in automating ER is the

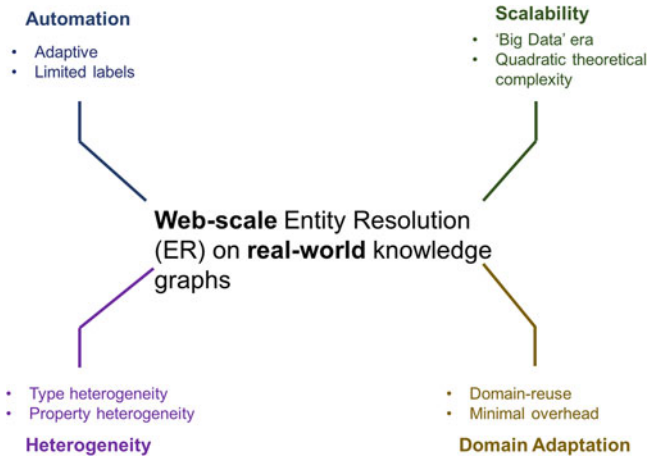

Fig. 3.2 An illustration of challenges that must be fulfilled to resolve entities in real-world knowledge graphs. Note that, even when graphs are not 'Web scale' a scalability challenge still arises because of the quadratic theoretical complexity of ER

ambiguity of the information extracted. Ambiguity is much harder to resolve in the presence of noise, and without access to underlying text, but even after accounting for those, it is not completely obvious how a machine is supposed to figure out that 'Nadal' in the first document refers to 'Rafael Nadal' or 'Toni Nadal'. In the much harder version of the problem, one would also have to figure out that 'Hamburg' in the first document is actually referring to the 'Masters Series Hamburg' and not the location Hamburg, as in the last document. More generally, because of nicknames (e.g., Pistol Pete vs. Pete Sampras) and other alternatives, one may also have a hard time coming up with viable candidates unlike the previous two cases, where one is faced with the finer-grained problem of choosing among viable linking candidates for a given node. Additionally, as in the natural language version of the problem, there is also the issue of having to deal with singleton nodes i.e. those that show up only once in the corpus and have no links to any other nodes.

Perhaps the most important challenge for an AI system attempting to counter the ambiguity in ER is that humans seem to draw on *background*, often intuitive, knowledge (often without even conscious reasoning) in several common ER domains that can be hard to pin down precisely in code. A computational challenge that will become more apparent in the subsequent discussion is *scale*, since naïve solutions to ER grow quadratically with the number of nodes in the KG [42]. As the introduction also pointed out, multi-schema ER is still very much in the nascent stages of research compared to the deduplication and single-schema cases. For knowledge graph construction systems that control the underlying reference ontologies of information extraction systems, multi-source, single-schema ER is still often applicable. However, in the most general case, multi-schema ER is necessary when constructing knowledge graphs over many sources, documents and

tables, and across the outputs of multiple (not often transparent) systems. Building such an ER system is still a challenge, especially if the domain is unusual in some way and there is little guidance by way of prior work in academia, or precedent in industry. Good performance from machine learning-based ER systems (the state-of-the-art) may also require a lot of training data, which is hard to acquire, since regular sampling and annotation does not work well in ER due to *data skew* (intuitively, this can be understood by considering that one entity, if randomly paired with another entity, will almost always never be a duplicate pair). While modern machine learning techniques can *partially* deal with the challenge of limited labeled data, human-level performance is yet to be achieved in the general case, and data augmentation, transfer learning and semi-supervised learning techniques from other machine learning applications (and theory) have yet to make a strong mark on ER. Finally, noise in the input, usually because of the imperfections of IE systems but also due to incompleteness in the original data, also have to be dealt with, since KGs are rarely constructed over data sources that are already easy to reason with.

Given these challenges, it should not be surprising that real-world ER systems perform well in *some* aspects, such as automation or scalability, but may be deficient in others (e.g., heterogeneity) [85, 86]. Figure 3.2 captures these requirements visually.

Automation First, given the increasing expense of data scientists and subject matter experts, an ideal ER solution should exhibit a high degree of automation. This requirement *can* be met by a non-adaptive system, but such a system would have low robustness or real-world utility. If the system is adaptive and uses some form of machine learning, the requirement can only be fulfilled by algorithms that are minimally supervised (i.e. use small amounts of training data) or more rarely, completely unsupervised. An alternate option that has been explored in industrial ER is to leverage crowdsourcing or a professional annotation service. This option is limited by both cost and scalability.

Scalability The size and growth in data ecosystems like Wikipedia, social media, Linked Open Data, webpages, sensor data and schema.org markup (Chap. 5) suggests that building a feasible ER system requires devising solutions that meet requirements of elastic scalability, preferably requiring computational resources that increase only *linearly* in the size of the data. While for many algorithmic pipelines, this is an achievable goal, it is much harder for ER. The reason is that ER is inherently (and theoretically) *quadratic* as we subsequently describe. Bringing down this quadratic complexity to *almost-linear* complexity is a field of research in its own right (called blocking).

Heterogeneity Earlier, we already suggested that multi-schema ER is becoming more important for KG-centric applications. For the purposes of ER, multi-schema heterogeneity can be broken down into two separate (but inter-related) problems. The first is *type heterogeneity*, which arises when different ontologies are used for different raw data elements. For example, one IE system may produce fine-grained

types such as *Inventor* and *Politician*, while another may produce coarser-grained types (e.g., *Employed Person* and *Unemployed Person*). The problem is further compounded by potential noise in type annotations, and by the presence of overlapping but not perfectly aligning type hierarchies across different sources and IE sub-systems. For example, is an inventor employed? The problem is more common than it seems, since the ontologies of many domains and datasets are developed relatively independently. Except in a few domains (such as the Gene Ontology in the biology domain [7]), heterogeneous ontologies and type-sets are the norm, rather than exceptions.

The second heterogeneity problem is *property heterogeneity* (the matching of property or edge labels across ontologies) that tends to arise once types are aligned. For example, let us assume that an ER system has correctly managed to address type heterogeneity by aligning *Inventor* (in one ontology) with *Entrepreneur* (in a second ontology). The ER system would also have to deduce such alignment relationships between properties such as *:co-founder_of* and *:organization*. As these examples show, alignment does not necessarily imply relationships of subsumption or equivalence, but is simply an empirical determination of sufficient entity overlap. Just like other processes in KGC, like IE, *instance-driven ontology alignment* is itself a problem that continues to be researched and has not been solved with human-level performance [1].

Domain-adaptation Finally, if the ER system is to be *re-used* across domains, it must also be *domain-adaptable* in its workings. By domain-adaptable, we do not mean that the ER system has to be a static one-size-fits-all model that magically works well across all domains, or even that it needs to be trained in one domain but is expected to perform well in a separate test domain (transfer learning). Rather, it must have the ability to adapt as the domain changes. In this sense, domain-adaptability is not necessarily mutually exclusive from domain-specificity, but refers to the *meta-ability* of an ER system to be re-trained, re-deployed and re-used on a different domain with *minimal overhead*. Domain-adaptability is hard to formalize; it is a practical and empirical constraints. In practice, no ER system is completely domain-adaptable (some assumptions built into the system are directly influenced by a use-case) or completely domain-specific (some re-use is always possible, and more re-use is generally possible in related domains). However, some systems are so strongly influenced by a particular use-case (e.g., product Entity Resolution) that adapting them to other domains is equivalent to writing the system from scratch. Event resolution is emerging as an excellent example of this phenomenon. Although event resolution is still heavily in flux as a research area, with a growing body of output, the best systems (both for event resolution and extraction e.g., BBN ACCENT [153]) tend to be heavily tuned not only for events, in general, but specific *types* of event. It is not unreasonable to suppose that an event resolution system designed for geopolitical events may not do as well if transferred to concert or entertainment events. Characterizing and evaluating such transferability is currently an open research problem.

In summary, there is a natural tradeoff between domain-adaptability and automation, and the two tend to influence each other in the design phase. For domain-specific ER systems, it is unlikely that (even with reasonable training data) the system will be able to resolve entities that look very different from the entities the system was designed to resolve. Modern *representation learning* techniques, such as word and graph embeddings [90, 132], have alleviated concerns about domain adaptation to a certain extent, since embeddings can be trained on unlabeled corpora. There is no free lunch however, since training good embeddings requires a sufficiently large corpus. In some domains, availability of such corpora may be limited.

3.3 Two-Step Framework

Even in early research, the quadratic complexity of pairwise ER was well recognized [123]. Given two data sources G_1 and G_2, where the set of non-literal entities in graph G is represented by the symbol E, a naïve ER system would evaluate all possible entity pairs. Assuming constant cost per evaluation, the run-time would be $O(|E_1||E_2|)$. In the rest of this section, for two entity sets E_1 and E_2, an entity pair (e_1, e_2) is denoted as bilateral iff $e_1 \in E_1$ and $e_2 \in E_2$. Given a collection of entities from $E_1 \cup E_2$, two entities e_1 and e_2 are said to be bilaterally paired iff (e_1, e_2) is bilateral.

To mitigate the quadratic complexity of generating all possible bilateral pairs, a two-step approach is adopted, as illustrated in Fig. 3.3 [41]. The first step, *blocking*, uses a many-many function called a blocking key to cluster approximately similar entities into overlapping blocks [42]. Only entities sharing a block are bilaterally paired and become candidates for further evaluation by a link specification function in the similarity step [172]. The link specification function may be either Boolean or probabilistic, and is used to indicate whether a candidate entity pair represents the same underlying entity.

Because ER developed as an important research area in the database community, the majority of ER research still assumes input databases to be structurally homogeneous i.e. if more than one database is input to the ER system, the databases

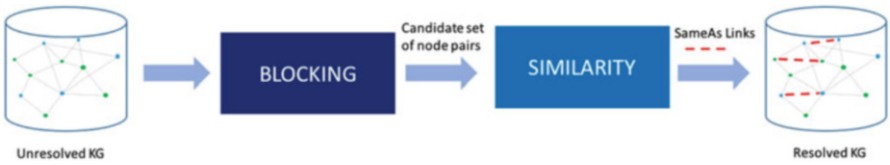

Fig. 3.3 The typical two-step workflow adopted for Entity Resolution

are assumed to have the same schema and same semantics [41, 57]. In the knowledge graph world, this would be equivalent to matching entities between knowledge graphs that have the same underlying ontology i.e. sets of concepts and properties. An important special application of structural homogeneity is deduplication, whereby matching entities in a single dataset must be found. Although structural homogeneity may seem like a limitation (which in some applications, can be severe), it is also often the case that ER is the next step after information extraction in a domain-specific KGC pipeline, and a single ontology is involved. Thus, the goal of ER is to deduplicate sets of entities extracted and typed according to this ontology. In the rest of this section, structural homogeneity is assumed. Later, we will briefly discuss extending the two-step model to include structural heterogeneity, but for an extensive discussion refer the interested reader to [86].

3.3.1 Blocking

Blocking is a preprocessing step that is used to mitigate the quadratic complexity of applying the link specification function on all (unordered) pairs of mention nodes in the knowledge graph. Given a set M of mention nodes (i.e. 'raw' entities extracted from documents), this *exhaustive set* contains $(|M||M| - 1)/2$ distinct unordered pairs, which is an untenable number of link specification computations for $|M| \gg 1000$. In the most general case, blocking methods use a many-many function called a *blocking key* to cluster approximately similar entities into overlapping blocks.

Definition 3.1 (Blocking Key) Given a set M of mention nodes, a blocking key K is a many-many function that takes a mention $m \in M$ as input and returns a non-empty set of literals, referred to as the blocking key values (BKVs) of m.

Let $K(m)$ denote the set of BKVs assigned to the mention $m \in M$ by the blocking key K. Furthermore, without loss of generality, the literals in the definition above are all assumed to be strings.

Example 3.1 (Blocking Key) Assuming the publication domain, with the *Publication* concept being the domain of properties Author, Venue and Year, and with the special property : *label* indicating the title of the publication, one possible blocking key K for deduplicating citations might be $overlap(Author(m_1), Author(m_2)) \wedge commonToken(Venue(m_1), Venue(m_2))$. This rule says that two publication mentions should share a block if their titles have at least three common tokens, *or* their venues have a common token (e.g., ACM KDD vs. KDD). We return to this example later in the context of automatically 'learning' good blocking keys. Note that this blocking key can generate multiple blocking key values for each node. More precisely, if a mention node has j authors and t tokens in its venue, the number of possible BKVs for the node is $j + t$. If any of these $j + t$ BKVs intersect with the BKV set of another node, they would fall within the same block (labeled by its BKV). The two nodes would share more than one block if they share more than

one BKV. Intuitively, this would happen when they share more than one common token across their venue attributes, or they share more than one author, or they share an author *and* a common token in their venue attributes (or any combination of the three options). For reasons covered shortly, some blocks may end up being discarded. In many situations, therefore, the higher the number of shared blocks between two mentions, the higher the probability they will actually be compared in the similarity step.

Given a blocking key K, a *candidate set* $C \subseteq M \times M$ of mention pairs can be generated by a *blocking method* using the BKVs of the mentions. We describe three influential methods that are generally included in established surveys [42], and all of which assume that a blocking key K is already specified by a user. Depending on the method, K must also obey some constraints. Subsequently, we also describe the automatic *learning* of good blocking keys.

3.3.1.1 Traditional Blocking

Given a blocking key K, an obvious solution is to generate the candidate set C as the set $\{(m_i, m_j) | m_i, m_j \in M \wedge m_i \neq m_j \wedge K(m_i) \cap K(m_j) \neq \{\}\}$. Put simply, if two mention nodes share a blocking key then they would be paired and inserted into C. Note that the definition of C as a set further implies that m_i and m_j may share multiple BKVs, although in the earliest definitions of this so-called *traditional blocking*, a mention was allowed to have exactly one BKV (hence, blocks could not overlap but represented a partition, with singleton blocks automatically discarded from further comparison).

A problem with traditional blocking approach is that of *data skew*. Consider, for example, two mentions from a *People* knowledge graph that are blocked based on the tokens in their last names. Last name frequencies in many countries tend to exhibit skew (a Zipf-like distribution) for some values (e.g. *Smith* in English-speaking countries). A consequence of the skew is that the run-time of the blocking method ends up being roughly proportional to the number of pairs generated by the largest block. This implies that run-time is still roughly quadratic in the number of mentions, unlike state-of-the-art blocking methods, where run-time tends to be slightly super-linear.

Despite this problem, traditional blocking is often the first line of attack in practical systems. In recent years, researchers have modified traditional blocking to handle the large blocks that result from skew. A simple method that is easy to implement and difficult to outperform is *block purging*. The premise of the method is that, with a sufficiently expressive blocking key, blocks that are too large can be safely ignored. Such blocks are most likely indexed by BKVs that are equivalent to stop-words like *the* or *an*. The algorithm takes a purging threshold as an input parameter, and discards all blocks that have more pairs than this threshold. The threshold may be learned from the data, and tends to be empirically robust to good default values as long as the default value is not too low.

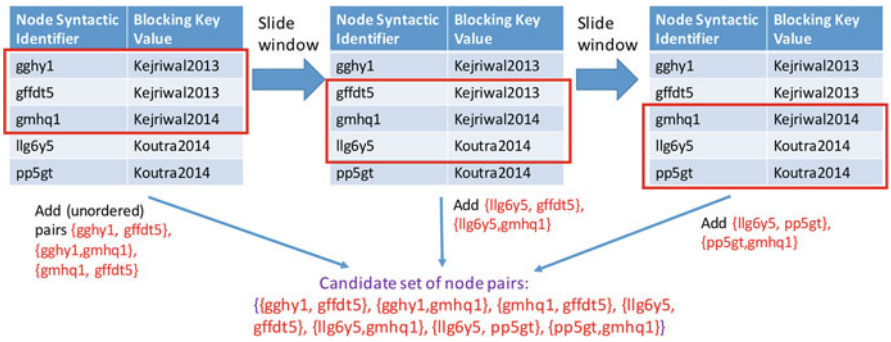

Fig. 3.4 An illustration of the Sorted Neighborhood workflow

3.3.1.2 Sorted Neighborhood

Another influential blocking method that was fundamentally designed to *guarantee* a bound on the size of the candidate set is the Sorted Neighborhood (SN) method, also known as *merge-purge* [76]. The algorithm works as follows. First, a single blocking key value (BKV) is generated for each mention using a many-*one* blocking key. Next, the BKVs are used as sorting keys to impose an ordering on the mentions. Finally, a window of constant size w is slid over the sorted list. All mentions sharing a window are paired and added to the candidate set. Figure 3.4 illustrates a workflow with a sliding window of size 3. We assume that the single BKV is generated by concatenating the last name of the first author of the publication with the year of the publication.

The sliding window has two implications for candidate set generation. First, mentions with *different* blocking key values may *still* get paired. This happens when the window straddles mention IDs in the list that have consecutively sorted BKVs (e.g., $gffdt5$ and $llg6y5$ get paired in Fig. 3.4). Second, some mentions with the *same* blocking key value may *not* get paired. For example, in Fig. 3.4, if the BKV for node $gmhq1$ had been Kejriwal2013 instead of Kejriwal2014, and the window size had been 2 instead of 3, then the node pair $\{gghy1, gmhq1\}$ would not be added to the candidate set.

Assuming that the window size w is much smaller than the total number of mentions, Sorted Neighborhood has time and space complexity that is *linear* in the size of the data. For this reason, it has endured as a popular blocking technique, especially when inputs are highly structured and it is possible to devise good blocking keys that yield a single, reliable BKV per mention. Numerous variations now exist, including implementations in Big Data architectures like Hadoop and MapReduce [95]. In general, the primary differences between the variants and the original version are input data types (e.g., XML Sorted Neighborhood vs. Relational), constraints on blocking keys and tuning mechanisms for the sliding window parameter (e.g. adaptive vs. constant) to achieve maximal performance in the similarity stage.

The main disadvantage of SN algorithms for KG-centric deduplication is their reliance on a single-valued blocking key. The authors of the original SN algorithm recognized this as a serious limitation and proposed *multi-pass SN*, whereby multiple blocking keys (each of which would still have to be single-valued) could be used to improve coverage. For a constant number of passes, the run-time of the original method is not affected asymptotically. Practical scaling is achieved by limiting the number of passes to the number of cores in the processor.

However, because even in multi-pass SN, each blocking key still remains single-valued, the use of expressive blocking keys (or even simple token-based set similarity measures that have high redundancy) is precluded. Extending SN to account for heterogeneous data sources is also non-trivial. For this reason, the application of Sorted Neighborhood to knowledge graphs and other heterogeneous, semi-structured data sources has been limited. The use of a simple blocking method such as traditional blocking (combined with skew-compensating measures like block purging) has remained popular for that reason.

3.3.1.3 Canopies

Clustering methods such as *Canopies* have also been successfully applied to blocking [112]. The basic algorithm takes a *distance function* and two threshold parameters $tight \geq 0$ and $loose \geq tight$, and operates in the following way. First, a seed mention m is randomly chosen from M. All mentions that have distance less than $loose$ are assigned to the *canopy* represented by m. Among these mentions, the mentions with distance less than $tight$ (from the seed mention) are removed from M and not considered further. Another seed mention is now chosen from all mentions still in M, and the process continues till all points have been assigned to at least one canopy.

In the Canopies framework, each canopy represents a block. However, unlike more typical methods like Sorted Neighborhood, Canopies does not rely on a blocking key, and instead takes a distance function as input. For this reason, at least one work has referred to it as an 'instance-based' blocking method, and distinguished it from 'feature-based' blocking methods such as Sorted Neighborhood and Traditional Blocking.

Similar to other popular blocking methods like Traditional Blocking and Sorted Neighborhood, several variants of Canopies have been proposed over the years, but the basic framework continues to be popular. For example, a nearest-neighbors method could be used for clustering mentions, rather than a threshold-based method. In yet another variant, a blocking key can be used to *first* generate a set of BKVs for each mention, and Canopies can then be executed by performing distance computations on the *BKV sets* of mentions, rather than directly on the mentions themselves. Because this variant relies on a blocking key, it can no longer be considered an instance-based blocking method.

For the distance function, the method has been found to work well with (the distance version of) a number of token-based set similarity measures, including Jaccard and cosine similarity [12], but in principle, many other distance functions can be used.

3.3.1.4 Research Frontier: Learning Blocking Keys

Earlier, we presented an example of a blocking key $overlap(Author(m_1), Author(m_2)) \wedge commonToken(Venue(m_1), Venue(m_2))$. This key, while intuitive, has some severe drawbacks. First, it would cluster together all papers authored by the same author into one single block. Some authors have many hundreds of papers, and some others collaborate with others who also have hundreds of papers. It is quite likely that, by itself, a rule such as this would end up placing a large number of publication mentions in a single block, which would negate the complexity benefits of blocking. A similar problem occurs with the second part of the rule, which says that a common token in venues is all that's required for two publication mentions to get blocked together. Tokens like ACM and IEEE are very common in venue titles (at least in Computer Science and Engineering), and once again, we would face the problem of having far too many mentions (the vast majority of which are non-matching) placed in one block.

In general, we note that the problem of *data skew* cannot really be avoided, unless the rules are extremely precise. Because there is a tradeoff between precision and recall in most real-world AI systems, the complexity reductions entailed by blocking would end up having a high cost in terms of lost recall. The goal of blocking always is to try and reduce complexity with minimal loss in recall. The blocking methods do provide some respite from data skew, if tuned correctly. For example, block purging would remove blocks that have too many mentions from further consideration in the similarity step. However, the problem with the blocking rule above is that a 'big' block would also contain many matching pairs along with non-matching pairs of mentions. Removing a big block would reduce complexity, but would yield recall that is almost trivially low.

This argument shows that, even with good blocking methods in place, the *quality* of the blocking key itself is very important for achieving a good tradeoff between recall and complexity reduction. Devising such a blocking key was once the turf of domain experts and knowledge engineers (and in many domains, still is), but with the advent of machine learning, it has been found that good rules can be learned automatically using a training set of labeled duplicate and non-duplicate mention node pairs.

The general idea is to frame the problem as that of learning rules in *Disjunctive Normal Form* (DNF). DNF formulae can technically be used to represent any propositional formula, but in practice, some restrictions are imposed (e.g., negations

of blocking predicates are not allowed, since this could make the blocking step intractable). The optimization function can be informally stated as that of learning a DNF formula such that: (1) the formula yields True for pairs of nodes in the positive training set, and (2) the formula yields False for pairs of nodes in the negative training set. It turns out that this problem can be decomposed as the famous *Red-Blue Set Covering* problem, which is known to be NP-hard. However, the problem is well-explored in the literature, and even a relatively greedy approach offers some good guarantees. In the few (but still growing) literature on blocking key learning, greedy algorithms were used to solve the reduced version of DNF blocking key learning. For more details, we refer the reader to fairly recent work in [87, 88] and [148].

3.3.2 Similarity

Once obtained, the candidate set C of mention pairs must undergo similarity computations to determine, whether probabilistically or deterministically, the subset of C that comprises duplicate mention pairs. In an i.i.d (independent and identically distributed) formalism, each mention pair can be independently assigned a score, with higher scores indicating greater likelihood of the pair being a duplicate pair. Although scores are typically normalized so that they lie between [0, 1], there is controversy about interpreting them as probabilities. We sidestep this controversy by continuing to refer to these numbers as scores.

Two issues now remain, one of which is concerned more with practice, and the other with theory. First, what methods do we use to obtain the scores in the first place? Intuitively, the 'goodness' of every such method should be measured by comparing against the perfect outcome i.e. the ground-truth. An ideal method that has knowledge of the complete ground truth would assign a score of 1.0 to every duplicate pair, and 0.0 to every non-duplicate pair. Subsequently, we present some formal methods for measuring performance using this principle. In practice, a validation or development set of labeled pairs can be used to select and tune methods to yield (by way of expectation) the best performing distribution of scores.

Second, given that scores output by a practical method will lie between 0.0 and 1.0, and will not necessarily be binary, how should one use these scores to 'partition' the set C into sets of duplicate pairs and non-duplicate pairs? There is a well-known theoretical model in the early ER literature known as the *Fellegi-Sunter model*, named after the scientists who first formulated it. The model generally requires not one but *two* (not necessarily distinct) thresholds to achieve a desired optimal tradeoff between the often conflicting goals of minimizing false positives and false negatives (which affect both *precision* and *recall*; see Sect. 3.4). The intuition behind using two thresholds is that they partition the set of mention pairs into three sets (matches, non-matches and *possible* matches i.e. pairs requiring manual review). To compute the score that will be compared to these thresholds, the ratio of conditional probabilities (with the condition based on whether the pair if

assumed to be matching or non-matching) is used. For more details on the Fellegi-Sunter model, we refer the reader to [61].

We also note that, for models that rely on rule bases or heuristics, labels may be directly output. However, to get good rules or heuristics, extensive domain engineering effort is required and in recent years, such methods have been largely superseded by machine learning. Therefore, we focus on machine learning methods for assigning scores to pairs in the candidate set. The evolution of the ER field (not necessarily within knowledge graphs or Semantic Web alone) is complicated; we provided an extensive survey, and the limitations of existing work, in [86].

In machine learning-based ER, each mention pair in C is first converted to a numeric (typically, but not necessarily, real-valued) *feature vector*. Figure 3.5 illustrates the procedure for two mentions, assuming that special (i.e. 'dummy') values are used in the event that (1) values for a given property are missing from both mentions, even though values for that property were observed for at least one other mention in the dataset; (2) the value for a given property is missing from one (but not both) mentions.

In general, given n properties, and m functions in the feature library, the feature vector would have mn elements. We say general, because it is also possible that some features are designed for specialized values (e.g., a feature that computes the number of milliseconds between two date values), and not applicable to two

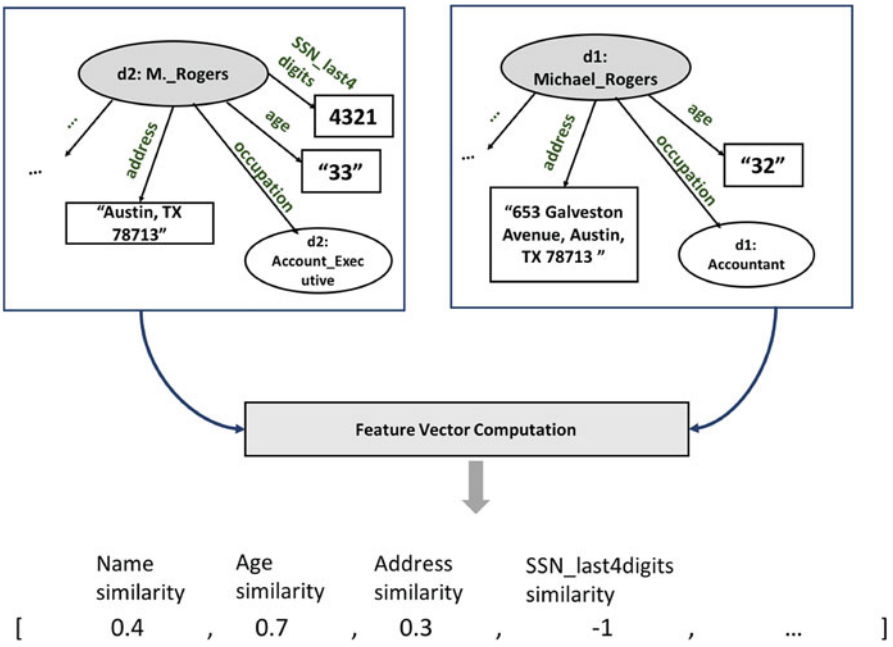

Fig. 3.5 An illustration of feature vector computation (between the two nodes mentioning Michael Rogers) assuming structural homogeneity. −1 is the dummy value used in this example

arbitrary values. However, having so many features, many of them correlated and often not useful, can be detrimental to machine learning generalization, especially when the training set is small and highly heterogeneous, as is the case with real-world ER tasks. There are several remedies for this; we consider two of the most popular ones. First, one can start by computing all possible (i.e. mn) features and then apply a feature selection method like Lasso. Second, one could spend some amount of domain engineering effort assigning only a few (in many cases, one might be sufficient) features to each property. Assuming that at most c feature function are assigned to a column, with $c \ll m$, the total length of the feature vector will be much less than mn, which would lead to presumably faster generalization and more robust performance. Often, domain expertise can be leveraged to limit m to 1 by deciding which feature function might be best for a given property. Considering again the example in Fig. 3.5, we may have decided to use NYSIIS for computing name similarity, normalized age difference for computing age similarity, TF-IDF for computing address similarity and so on, as opposed to applying every single function in our library for every possible property feature computation. Intuitively, one would not want to apply NYSIIS to address similarity since it would likely not be useful (and may even cause noise and problems with generalizing on fewer training examples, an important concern).

What feature functions should be included in a library? There is an enormous body of work on both string similarity, and to a lesser degree, phonetic similarity functions, and not nearly as much research on numeric or date types. Software packages in multiple languages exist that implement many of these similarity functions. For the sake of completeness, we provide a list of functions that have been popularly used in Table 3.1.

There is another problem that we alluded to at the beginning of the chapter, namely, what should we do when there are multiple values per mention per property? The pre-dominant way to extract features from such pairs of 'sets' is to consider a two-layer similarity function, where the first layer consists of an atomic similarity function (e.g., if the set consists of string values, this could be the normalized similarity version of the edit distance function), and the second layer consists of an aggregation. Aggregation functions of this kind have been explored in detail in the clustering literature. Below, we portray such a two-layer function using an example.

Example 3.2 (Two-layer Similarity Feature) Consider two sets of names {Jim, Jimmy, Jeremy} and {James, Jim} between which we need to output a single simi-

Table 3.1 Illustrative instances of similarity functions typically used in ER workflows. Neither the feature categories nor the example functions in each feature category are exhaustive

Feature category	Example functions
Character	Edit, Levenstein, Affine Gap, Smith Waterman, Jaro, Q-gram
Token	Monge Elkan, TF-IDF (Soft, Q-gram), Jaccard
Phonetic	Soundex, NYSIIS, ONCA, Metaphone, Double Metaphone

larity score. An atomic similarity feature could be Monge-Elkan. Namely, we could compute Monge-Elkan scores between each pair of terms in the two sets and output the results as a *complete weighted bipartite graph*. Several aggregation measures could be considered for the second layer of the similarity feature. For example, we could take either the minimum or maximum of all scores in the graph as the final score. We could also take the average. A more robust mechanism that has been found to work well in several cases is the Hungarian algorithm, which tries to match each term in the first set with at most one term in the second set (vice versa) such that the total sum of scores is maximized. We only keep those edges in the graph that were included in the optimal matching output by the Hungarian algorithm. Note that the number of edges in the graph will be the minimum of the cardinalities of the two sets, since a term in any set can never receive more than one assignment. In this sparser graph, containing only optimal assignments, we could take the average, minimum or maximum (or any reasonable aggregation) of edge weights.

As the example above shows, the more complicated a similarity measure becomes, the more degrees of freedom it tends to have, and the more options there are to explore. The extensive literature on ER is a good place to look for defaults, but for unusual domains, there is no substitute for careful tuning, some of which may have to be done through a systematic process of trial and error. Lately, vector space embeddings (covered in the next chapter) have alleviated some of the feature engineering effort that has gone into a typical ER workflow, but there is still much more work to do on this front.

Once a feature extraction methodology is in place, each mention pair in C is, in turn, converted to a feature vector. A machine learning classifier is trained on positively and negatively labeled training samples, and is thereby used to assign scores to the vectors in the candidate set. Several classifiers have been explored in the literature, with random forest, multilayer perceptron and Support Vector Machine (SVM) classifiers all found to perform reasonably well. We note that, although all of these classifiers make the i.i.d. (independent and identically distributed) assumption, transitivity does play a strong role in real-world ER determinations (if (e, j) and (e, k) are classified with high scores, it is reasonable to suppose that so should (j, k)). This fact is typically employed, not at this stage, but in the post-processing clustering and soft transitive closure stage (briefly discussed in a subsequent section) where we take the outputs of similarity and attempt to 'collapse' them into clusters, with each cluster representing all mentions of a single underlying entity.

3.4 Measuring Performance

The independence of blocking and similarity suggests that the performance of each can be controlled for the other in experiments. In the last decade, in particular, both blocking and similarity have become increasingly complex. It is the norm,

rather than the exception, to publish either on blocking or on similarity in an individual publication. Despite its potential disadvantages (in practice, there are interdependencies between blocking and similarity, since feature functions and biases could often be traded between the two, sometimes without knowledge), this methodology has resulted in the adoption of well-defined evaluation metrics for both blocking and similarity. This independence assumption has been challenged in a small number of applications in recent years; as just one example, a blocking technique called *comparisons propagation* proposes using the outcomes in the similarity step to estimate the usefulness of a block in real time [137]. The premise is that if a block has produced too many non-duplicates, it is best to discard it rather than finish processing it. By this logic, the cost of processing the block outweighs the gain, at least in expectation.

While such techniques are appealing, their implementations have mostly been limited to serial architectures, owing to the need for continuous data-sharing between the similarity and block generating components. Experimentally, the benefits of such techniques over independent techniques like Sorted Neighborhood or traditional blocking (with skew-eliminating measures such as block purging) have not been established extensively enough to warrant widespread adoption. The two-step workflow, with both steps relatively independent, continues to be predominant in the vast majority of ER research. With this caveat in place, we describe these metrics below.

3.4.1 Measuring Blocking Performance

The primary goal of blocking is to scale the naïve one-step ER that pairs all mentions (order-independently) with each other. A blocking system accomplishes this goal by generating a smaller candidate set. If complexity reduction were the *only* goal, the blocking system could simply generate the empty set and obtain optimal performance. Such a system would be useless because it would generate a candidate set with zero duplicates coverage.

Thus, duplicates coverage and candidate set reduction are the two goals that every blocking system seeks to optimize. To formalize these measures, let Ω be denoted as the *exhaustive set* of all ${}^{|M|}C_2$ pairs; in other words, the candidate set that would be obtained if there were no blocking. Let Ω_D denote the subset of that contains all (and only) matching mention pairs (i.e. semantic duplicates). Ω_D is designated as the ground-truth or gold standard set. As in previous sections, let C denote the candidate set generated by blocking. Using this notation, *Reduction Ratio (RR)* is defined by the equation below:

$$RR = 1 - \frac{|C|}{|\Omega|} \tag{3.1}$$

The higher the Reduction Ratio, the higher the complexity reduction achieved by blocking, relative to the exhaustive set. Less commonly, RR can also be evaluated relative to the candidate set C_b of a *baseline* blocking method (by replacing Ω in Eq. 3.1 with C_b). Note that, since RR has quadratic dependence, even small differences in RR can have an enormous impact in terms of run-time. For example, if Ω contains 100 million pairs (not an unreasonable number, since it would only take a mentions set M with about 20,000 mentions i.e. a relatively small dataset), and System 1 achieves an RR of 99.7%, while System 2 achieves 99.5%, their candidate sets would differ by 200,000 pairs.

In a similar vein, coverage, or Pairs Completeness (PC), is defined below:

$$PC = \frac{|C \cap \Omega_D|}{|\Omega_D|} \tag{3.2}$$

One interpretation of PC is to consider answering the following question: *if we knew $\mathscr{L}$ and apply it to the candidate set C output by blocking, what would be the recall?* From this perspective, PC is nothing but a measure of recall (used for evaluating overall duplicates coverage in the similarity step, as described in the subsequent section) that *controls* for the errors in further learning or approximating $\mathscr{L}$, which is *not* known. In other words, Pairs Completeness gives an *upper bound* on the recall metric. For example, if PC is only 80%, meaning that 20% of the duplicate pairs did not get included in the candidate set, then coverage on the full ER task will never exceed 80%.

There is typically a tradeoff between achieving high PC and RR. The tradeoff is achieved by tuning a relevant parameter. There are two ways to represent this tradeoff. The first is a single-point estimate of the *F-Measure*, or harmonic mean, between a given PC and RR:

$$F - Measure = \frac{2 \times PC \times RR}{PC + RR} \tag{3.3}$$

A single-point estimate is only useful when it is not feasible to run the blocking algorithm for multiple parameter values. Otherwise, a more visual representation of the tradeoff can be achieved by plotting a curve of PC vs. RR for different values of the parameters.

Another tradeoff metric, Pairs Quality (PQ), is less commonly used than the F-Measure of PC and RR:

$$PQ = \frac{|C \cap \Omega_D|}{|C|} \tag{3.4}$$

Superficially, PQ seems to be a better measure of the tradeoff between PC and RR than the F-Measure estimate, which weighs RR and PC equally, despite the quadratic dependence of the former. In this vein, PQ has been described as a precision metric for blocking. Intuitively, a high PQ indicates that the generated blocks (and by virtue, the candidate set) are dense in duplicate pairs.

In practice, PQ gives estimates that are difficult to interpret, and can be misleading. For example, suppose there were 1000 duplicates in the ground-truth, and only contained 10 pairs, of which 8 represent duplicates. PQ, in this case, would be 80%. Assuming that the exhaustive set is large enough that RR is close to 100%, the F-Measure (as defined above) would still be less than 2% (since PC is less than 1%). The F-Measure result would be correctly interpreted as an indication that, for practical purposes, the blocking process has failed. The result indicated by PQ alone is clearly misleading, suggesting that, as a tradeoff measure, PQ should not be substituted for the F-Measure of PC and RR. An alternative, proposed by at least one author but (to the best of our knowledge) not used widely, is to compute and report the F-Measure of PQ and PC.

3.4.2 Measuring Similarity Performance

Given a candidate set C, the similarity step uses a learned linking function to partition C into sets C_D and C_{ND} of duplicates and non-duplicates respectively. The two metrics predominantly used for evaluating the similarity step, and by virtue, ER as a whole, are *precision* and *recall*:

$$Precision = \frac{|C_D \cap \Omega_D|}{|C_D|} \tag{3.5}$$

$$Recall = \frac{|C_D \cap \Omega_D|}{|\Omega_D|} \tag{3.6}$$

In other words, precision is the ratio of true positives to the sum of true positives and false positives, while recall is the ratio of true positives to all positives in the ground-truth. Similar to PC and RR defined earlier, there is a tradeoff between achieving high values for precision and recall. An F-Measure estimate can again be defined for a single-point estimate, but a better, more visual, interpretation is achieved by plotting a curve of precision vs. recall for multiple parameter values.

Note that, since similarity is defined as a binary classification problem in the machine learning interpretation of ER, other measures such as accuracy can also be defined. One reason why they are not considered in the ER literature is because they also evaluate performance on the negative (i.e. non-duplicates) class, which is not of interest in ER. An alternative to a precision-recall curve is Receiver Operating Characteristic (ROC), which plots true positives against false positives. Historically, and currently, precision-recall curves dominate ROC curves in the ER community, but nowadays, important machine learning packages (e.g., sklearn in Python) allow a user to print out various metrics and curves without any programming. In real life, we recommend printing out both the precision-recall and ROC curves to evaluate both (1) how well the ER system is doing in an 'absolute' sense; (2) how well the ER system is doing above *random*.

3.5 Extending the Two-Step Workflow: A Brief Note

Although the vast majority of ER (including research and implementation) is concerned with optimizing and automating one or both of the two steps in the standard blocking-similarity workflow, the heterogeneity of knowledge graphs can require two additional steps to be given some thought in some application domains. Earlier in the chapter, we discussed how heterogeneous schemas (by way of type and property heterogeneity) can cause problems for ER systems. If we are linking mentions between two independent knowledge graphs with different schemas, or even between mentions in a single knowledge graph with very fine-grained types and properties, it is important to develop a robust type and property matching system that can be executed prior to blocking to reconcile schema heterogeneity. Just like blocking, we generally desire such preprocessing steps to be recall-friendly, since we do not want to risk losing (already sparse) duplicates before the similarity step, which is expected to add noise of its own.

It turns out that there is a large body of work on both type and property matching, sometimes involving the same researchers as ER, and generally falling under the umbrella term of *schema matching* or *ontology alignment*. In practice, simple solutions to ontology alignment get us most of the way in real-world domain-specific KGC pipeline, though advanced solutions are mandated if the accuracy requirements are higher than normal or if the domain is particularty difficult. More recently, there has also been some work on schema-free approaches that do not require an alignment between heterogeneous ontologies before executing an ER workflow. The efficacy of these approaches is not fully understood, however, since only a handful of papers have explored its applications.

Additionally, post-processing steps like clustering may also be required *after* the similarity step has finished executing. We mentioned clustering and transitive closure earlier, and these continue to be the most important post-processing steps.

3.6 Related Work: A Brief Review

Entity Resolution has been a research area for almost fifty years, even though the problem has picked up a large amount of steam only in the last couple of decades owing to the growth of the Web. Recall that we had listed four important challenges when we had first described the problem (i.e. automation, heterogeneity, scalability and domain adaptation). Concerning the last challenge (domain adaptation), we note that most ER solutions in research tend to be domain agnostic, although a few are specifically geared for customer (both people and business) names. Many of the best domain-specific ER systems (such as for product names, or for publications) tend to have been developed in industry, and likely required a lot of proprietary training, tuning and model engineering. Scalability efforts have tended to attract the attention of the database community (by way of devising the problem as optimizing ‘soft

joins') and to a lesser extent, parallel and distributed systems. There is a tradeoff, however, between automation and scalability, as we later discuss.

Our main focus in the discussion herein will be on automated solutions, with some focus on heterogeneity. The reason is that automation continues to be the most important issue, both in the broader AI community, but especially in problems like ER where intensive effort is usually required to achieve good quality.

3.6.1 Automated ER Solutions

Since the early 2000s, machine learning has been actively applied to ER [57]. A machine learning-based ER system could adaptively learn good blocking and similarity functions from both the labeled training data (for supervised approaches), and the unlabeled data (for unsupervised, semi-supervised and clustering-based approaches). On the other hand, systems that use a fixed set of heuristics on all data sources are non-adaptive, and by any pragmatic definition, the issue of automation trivially does not arise.

One of the earliest examples of an adaptive ER system, proposed by Winkler [183], uses a variant of the Expectation Maximization (EM) algorithm [53]. The Fellegi-Sunter model of record linkage is assumed [61]. In this model, candidate entity pairs are partitioned into three classes (matches, non-matches and possible matches) using two decision thresholds. The class of possible matches includes entity pairs that are too ambiguous for the similarity function to resolve into a match or non-match class. Such pairs require clerical review. A Bayesian argument shows that using two decision thresholds is optimal in the sense of minimizing possible matches for preset Type I and II error rates [61].

Unfortunately, Winkler [184] stipulated that the EM algorithm can only be successfully applied to ER if at least five empirical conditions are met. Elmagarmid et al. [57] succinctly list these conditions, some of which are problematic for Linked Data. One such assumption is conditional independence of features. Another is that the match class is well-separated from the non-match class. In systems were EM was considered as a baseline, the empirical performance was found to be less than ideal when some of the stated assumptions are violated [89].

Ravikumar and Cohen [150] use similar, but more robust, ideas by proposing hierarchical graphical models as a way of modeling the similarity of features through latent variables. The system is unsupervised, but assumes structural homogeneity and a serial architecture. A distance function[1] is also assumed to be provided. Empirically, the scope of the work was limited to Relational Database deduplication applications.

On a similar note, Bhattacharya and Getoor [20] use Latent Dirichlet Allocation (LDA) for modeling latent commonalities between entities [29]. The main appli-

[1] In the paper, Soft-TF-IDF was proposed as an excellent distance function [150].

cation of their work is in *collective* classification [21]. A classic example arises in the co-authorship domain. Given a set of bibliographic works, two authors (on two independent works) are likely to be the same individual if they have similar co-authors. By modeling such relational information through latent variables, pairs of individuals can be collectively disambiguated. While promising, the work has not been shown to be applicable to domains where relational issues don't arise. Similar to the work by Ravikumar and Cohen [150], structural homogeneity and serial execution were both assumed in the original paper [20].

Christen [39] adopts a different approach. First, a strong weight-based heuristic is used to sample training examples that are almost certainly matches or non-matches. Intuitively, the feature weights in such examples are nearly all 1.0 for matches (or 0.0 for non-matches). The method is predicated on locating such extreme-weighted samples to bootstrap the training process. A classifier (SVM) is trained on the samples and used to label other feature vectors in the candidate set. The method, along with other viable classifiers, a synthetic data generator and a user interface, is available in the FEBRL toolkit [40]. FEBRL was originally designed for biomedical record linkage (a much more constrained form of ER, pertaining primarily to the database community), but can be applied to other domains. Heterogeneity is a major issue, since FEBRL is designed for structurally homogeneous applications. Empirically, only small benchmarks were used for evaluations.

Systems based on active learning have also been proposed, two good examples being RAVEN and COALA [125, 128]. Such systems do not require as many training examples as fully supervised systems such as MARLIN [23], and deliver competitive performance. A major disadvantage is scalability, owing to the method being iterative and requiring continuous user participation. On a positive front, heterogeneity is less of an issue as these systems were designed for Linked Data applications. In particular, RAVEN accommodates structural heterogeneity by modeling type and property alignments as an application of the stable marriage problem [72].

Genetic algorithms have also been extensively explored [126], both in supervised and unsupervised versions. The unsupervised version relies on a measure known as a pseudo F-Measure (PFM). PFMs are heuristics that aim to approximate the actual F-Measure by analyzing the data, and are used as fitness functions in the genetic algorithms. A PFM can also be used to guide the unsupervised learning of a link specification function, as in the deterministic EUCLID algorithm, which uses linear and Boolean classifiers [127]. Although promising, evaluations have shown that the correlation between various proposed PFMs and the actual F-Measure is tenuous [127]. With genetic approaches, the entire dataset has to be scanned over multiple iterations, and results are non-deterministic. In the original papers, EUCLID and the genetic algorithms also did not include solutions for type and property alignments, and were evaluated on small benchmarks [126, 127]. Taken together, these observations indicate that these algorithms may not be suitable for large-scale Linked Data applications.

A promising solution that requires training data, but that can then be applied to other datasets with minimal supervision through transfer learning was proposed by

Rong et al. [159]. This solution is also one of the few to favor both automation and heterogeneity, the latter by virtue of employing schema-free features. An example of a schema-free technique that was earlier introduced in Chap. 2 was Canopies [112]. Such techniques address heterogeneity in a brute-force fashion, by ignoring all structural information. In the case of [159], features are extracted by jointly considering the information set of all properties (of a candidate instance pair). For example, a numeric parser is used to extract numeric information (e.g. dates) present in the properties. A problem with using such features is that noise can be introduced by extracting irrelevant information. Also, Rong et al. [159] do not directly address type heterogeneity. Finally, while transfer learning has some advantages, it also degrades occasional performance. Determining when to use transfer learning is an ongoing area of research [136].

3.6.1.1 The Automation-Scalability Tradeoff

There is a queer tension between automation and scalability. Extremely scalable ER systems, such as Dedoop, require user involvement in terms of specifying the workflow, as described below. Broadly speaking, it has been found that scalable systems tend to make strong assumptions. Locality Sensitive Hashing techniques, for example, assume that appropriate hashing families exist for the distance functions being approximated [52]. In literature covering both ER and ontology or schema matching, the only functions for which LSH has been appropriately utilized are Jaccard and a version of the cosine distance function [56]. An extension to LSH techniques to accommodate the properties of machine learning classifiers is by no means straightforward. Another example of an architecture amenable to parallel and distributed algorithms, Swoosh, also imposes strong assumptions on the similarity function [15].

It is also interesting to note that ER systems implemented in a shared-nothing paradigm, such as MapReduce, tend to leave the burden of specification on the user. We mentioned Dedoop earlier as an example that requires the user to completely specify the workflow [94]. The same is true for LIMES and SILK [124, 172], which are not implemented in MapReduce, but require the user to specify the appropriate functions and parameters. Smart joins and soft joins, which have witnessed much research in the database community, are not adaptive and generalizable in the way proper ER systems are.

A promising approach that is potentially amenable to a fixed number of approximately linear-time MapReduce jobs is the SVM-based proposal by Christen [39]. In its present form, the proposal accommodates neither scalability nor heterogeneity. The latter problem can be dealt with, as described in the following section. It is less obvious how the system can automatically and scalably locate good seed examples to bootstrap the training process. Christen makes the assumption that seeds can be unambiguously located by seeking feature vectors with weights that are nearly all 0 or 1. With noisy data, this is almost never guaranteed. In empirical findings, feature vectors are often found to be sparse, even for duplicates. If a potential method can

locate seeds from such data using a fixed number of MapReduce jobs automation and scalability requirements can be reconciled. Once located, seeds can be used, in principle, for training not just a machine learning similarity classifier, but also for learning DNF blocking schemes and determining property alignments.

It is also possible to survey this issue from the opposite end of the spectrum. Automated systems, which mainly tend to be EM-based algorithms that iteratively refine a likelihood function by learning good parameters for latent variables, require multiple scans over the dataset, copious amounts of data sharing and an unspecified number of iterations before convergence [20, 150]. In general, they are non-deterministic and may require multiple re-starts to avoid the pitfalls of local optima. As Winkler (1993) observed [183], various empirical conditions have to hold for such algorithms to be viable. Recent progress on this last issue has been promising, but is, by no means, a settled matter [159].

Generalizing even further, the problem arises because automated systems yield a similarity function that may be 'opaque' in that it is a black-box function that cannot be further analyzed or optimized. In such cases, the only option is to rely on a good blocking function, which may itself require manual intervention. We subsequently describe some efforts in the direction of discovering good blocking functions without manual supervision. There has been some work on this, but by and large, there is no workflow that is both unsupervised and that is ultra-scalable.

3.6.2 Structural Heterogeneity

A traditional assumption in the ER community is that datasets have been homogenized prior to executing an ER workflow [96]. In the tabular community, schema matching is assumed to have been performed a priori [41, 57]. In the Semantic Web community, ontology matching is assumed to have been performed a priori [63].

These assumptions would not be problematic if the schema and ontology matching problems were solved. In fact, research on them has been ongoing for many decades [60, 147]. In some cases, schema matching systems like Dumas assume that ER has been solved a priori[2] [24]. The argument is that, despite the progress in both ER and schema matching, it is misleading to assume that either problem has been solved perfectly.

The question is largely empirical. Is it sufficient to use classic, relatively simple, approaches to address type and property heterogeneity in the broader context of ER? Empirical results have shown that while type heterogeneity is amenable to classic approaches [88], property heterogeneity is not [89]. Insofar as the related work is concerned, only the RAVEN system properly[3] deals with heterogeneity, although empirical evaluations on this issue are limited [125]. Other systems, like the one

[2] Dumas relies on duplicates to match columns.

[3] That is, RAVEN addresses heterogeneity through alignments, as opposed to ignoring structure.

by Rong et al. [159], address heterogeneity by using schema-free features that ignore structural properties altogether. There is an empirical argument against such approaches, for well-structured RDF graphs, since one would be losing valuable structural information by adopting a purely schema-free approach in devising features (while simplistic, an analogy would be the process of 'getting rid' of columns in a table by concatenating all columns into one column, which would make the schema matching process trivial, since one would be matching records between tables with only one column each).

Note that the barrier to adopting a heterogeneous solution is conceptually simple to overcome, by pre-pending alignment modules to the basic two-step workflow illustrated earlier. Recent progress on this issue has been promising, especially in the context of blocking [137].

3.6.3 Blocking Without Supervision: Where Do We Stand?

We mentioned earlier that there has traditionally been a strong focus (in the ER research community) on the similarity step. An unfortunate consequence of the complexity of recent ER research is that researchers often ignore other aspects, such as blocking, in their exclusive focus on similarity or scalability. For example, both [150] and [20] use simple ad-hoc blocking keys in their experiments.[4] Scalable systems make more extreme assumptions. For example, Dedoop require both blocking and similarity steps to be precisely specified by a user as part of an ER workflow [94].

For the same reason that ignoring the effects of schema or ontology matching prior to ER is dangerous, the effects of blocking on the overall workflow should not be neglected. Before a contribution in 2013 [87], DNF blocking scheme learners were, at best, semi-supervised [33]. Evaluations conducted in prior work by the author show that clustering techniques such as Canopies do not work well on a variety of datasets [112]. In the Semantic Web, the only unsupervised blocking scheme learner, proposed by Song and Heflin in 2011 [166], was evaluated on small datasets and is not as expressive as general-purpose DNF blocking schemes. Thus, in real-world ER, unsupervised blocking cannot be assumed away, since it is not completely solved yet. Whether the community addresses this issue in sufficient detail, as opposed to a continuing skewed focus on the similarity step, will become clearer in the decade to come.

[4] For example, all records sharing a 4-gram character sequence were placed in the same block [150].

3.7 Summary

Entity Resolution is an important second step in a knowledge graph construction workflow following information extraction. The problem continues to be a difficult one, and has taken on renewed importance with the advent of knowledge graph ecosystems. Although the research has witnessed much progress, some issues are still outstanding. Particularly glaring is the lack of an adaptive, unsupervised ER system that learns from prior results, is amenable to domain adaptation and transfer and is scalable enough to deal with Web-scale graphs. Also lacking is a systems-level view of the problem, although in recent years, research has been catching up to industry in building end-to-end ER infrastructures for specific problem domains (such as products and geopolitical events).