Chapter 2
Information Extraction

2.1 Introduction

Information extraction (IE) is a fundamental component in any knowledge graph construction pipeline, whether domain-specific or generic [111, 119]. As the name implicitly suggests, the goal of an IE system is to extract *useful* information from 'raw' data, usually text or webpages. Useful information has many dimensions, but the most important dimension for computational purposes is that the information can be queried, and reasoned about, by machines. One of the reasons why IE was identified as an early problem in communities like natural language processing was because machines are not good at understanding natural languages like English or French due to problems like subtlety, ambiguity and irregularity. Even today, despite rapid advances, machines still cannot read and understand English nearly as well as humans. Thus, an early goal of IE was to extract key pieces of information, such as entities, relations, events and attributes from natural language text. The advent of the Web only made the problem more interesting, since webpages are visually intuitive (if rendered in an compatible browser), but in their raw HTML versions, contain many interesting 'markup' elements like tables, lists, links, images and even dynamic elements like Javascript programs [36].

In Chap. 1, we implicitly demonstrated the results of 'perfect' information extraction when we converted an English sentence (about Fido the dog) to 'equivalent' knowledge graph triples. To take another example, a sentence such as 'Vladimir Putin, President of Russia, attended the G20 meeting', can be represented by the following set of triples: {(Vladimir Putin, presidentOf, Russia), (Russia, is-a, Country), (Vladimir Putin, is-a, Person), (G20, is-a, Geopolitical Meeting), (Vladimir Putin, attended, G20)}.

Recall that we referred to elements like 'Vladimir Putin' as entities, while elements like 'presidentOf' are generally referred to as relations, relationships or properties. We also introduced the notion of literals (also called attributes or slot fillers depending on both the context and community) in the previous chapter; e.g., if we had a triple such as (Vladimir Putin, DOB, “10/07/1952”), the date of birth would be a triple. Generally, numbers and strings are understood to be literals.

The example above notwithstanding, the term ‘IE’ is far too broad to be useful in practice, since there is no one IE system that can extract all possible entities and relationships from a (‘completely triplify’) given English (or other natural language) sentence. Generally, IE systems are constrained by an underlying ontology, just like an actual knowledge graph, although in recent years, the concept of Open IE (which is open-world and not constrained by an ontology) has been gaining some traction [10].

In this chapter, we take a practical view of IE based on established literature. Over the last few decades, IE has been heavily researched and many techniques are currently in use in the community, including classic rule-based techniques, more modern sequence labeling techniques such as Conditional Random Fields (CRFs) [100] but also more cutting-edge techniques such as deep neural networks [47]. We begin by describing why IE is so challenging, and why it continues to be the most important component in building high quality KGs from scratch. Next, we survey IE from a range of functional perspectives.

2.2 Challenges of IE

IE has been explored in the AI community for several decades now [48, 161]. That the problem is still actively researched and has not been solved yet is a testament both to its difficulty and its relevance. In this section, we explore some predominant challenges that prevent IE systems from reaching arbitrarily high quality for many real-world datasets.

First, state-of-the-art IE systems tend to be based on supervised machine learning, a class of techniques whose success is predicated on having access to labeled training data. Labeling data from scratch is an expensive and time-consuming endeavor that cannot be effectively scaled. In some cases, the labeled data itself (as opposed to the algorithms trained on the data) is the basis for a system’s competitive advantage, and is closely guarded, especially in the commercial sphere. However, due to efforts like the Message Understanding Conferences (MUCs) [68], the research community as a whole has come a long way in developing a robust set of benchmarks both for training and evaluating IE algorithms. However, even within this body of labeled data, some IE tasks (like named entity recognition) are much better supported than others. As novel IE tasks and data emerge, such as joint text-video extraction and event extraction, the utility of previously labeled datasets become less clear [175].

Second, for domain-specific KGC, IE presents some additional challenges. Like with so many techniques that use machine learning and are optimized for a ‘generic’ (or open-world) domain such as the encyclopedic world covered by Wikipedia or the Google news corpus, problems especially arise in domains that are different from these common corpora and require special techniques for maximal performance.

The question that arises is: how does one build reasonably high-quality IE systems *without* access to a lot of labeled data? On a related note, how does one make *judicious* use of unlabeled data? What *kinds* of supervision are possible besides mundane labeling of lots of data? These questions have been explored in the research community for quite some time, and we cover some techniques in this chapter.

Third, the format and *heterogeneity* of the raw data is very important and can also be a challenge when transitioning results across communities or research groups. Are we extracting information from HTML webpages or from a plain text file? Within an HTML file, do we have a lot of tables, markup and even javascript? Isolating the relevant information from the page before running an IE system over it can itself be a challenge. We describe in a later section how wrapper induction techniques can be used to extract meaningful information from webpages [99].

2.3 Scope of IE Tasks

Because IE is such a diverse problem, any review must necessarily *scope* the problem. In some cases, the tasks are different enough that they get separated by community. Multiple surveys and reviews also tend to take this view; see for example [161] and [111]. Where possible, we attempt to follow a similar flow as others that have extensively surveyed individually components of this chapter. For example, much of the work on NER described in this chapter is closely inspired by the survey of Nadeau and Sakine [121]. For example, traditionally, it has been the case that Web IE was treated very differently from NLP-centric IE like extracting named entities, relations and events from text. Although these distinctions still remain, their significance has diminished, in part because even within each community a wide range of techniques and methodologies have flourished. For this reason, we do not separate Web IE and NLP-centric IE in this chapter, but consider them as different IE *tasks*. Among the different tasks below, the first three (Named Entity Recognition, Relation Extraction and Event Extraction) have been overwhelmingly researched by the NLP community. The last task, Web IE, has witnessed more research attention in the overall AI literature, with wrappers emerging as a dominant technique even in the early days of the Web. Machine learning has been used extensively for all the different IE tasks described below. Interesting combinations are also possible. For example, given a corpus of text-heavy webpages (e.g., blog articles scraped from an online portal), one may initially run a wrapper or Web IE system to strip out the HTML boilerplate, and obtain the underlying text. Next, a sufficiently trained and tuned IE for tasks such as NER and relation extraction may be applied.

In fact, many such combinations are possible, and the flexibility, architecture and quality of an overall IE pipeline depends significantly both on the experience and imagination of an application designer. The tasks below are not expected to be mutually exclusive, and some are mutually *reinforcing*. We illustrate a particularly important case of this when we describe joint event-entity extraction.

2.3.1 Named Entity Recognition

Named Entity Recognition (NER) is often the first line of attack in a given IE problem domain [121]. Given a document and a set $T = \{t_1, \ldots, t_n\}$ of n entity types (defined in an ontology, as previously covered), a NER system generally returns a set of *extracted mentions*, where each mention may be expressed in the form *(t, start-offset, end-offset)*, with $t \in T$. Consider the sentence '**Tom Cruise** shot the latest Mission Impossible movie in **Dubai**', where the extracted mentions are in bold. Given the type set {Person, Location}, the mention 'Tom Cruise' is clearly of type Person, while 'Dubai' is of type Location. It is precisely because the mentions are typed that they are often referred to as *named entities*. However, the term is also a misnomer, since mentions that refer to the same underlying entity need to be resolved to a single named entity. This process, called Entity Resolution (ER) in the graph-theoretic and broader KG community [66], will be covered in the next chapter. In a similar vein, the step of 'coreference resolution' is also often undertaken in the NLP community to leverage text and syntax features (such as part-of-speech tagging and Wikipedia linking [73, 173]) to link different mentions, including pronouns, to each other [122].

As the example sentence above illustrated intuitively, full named-entity recognition can be broken down into two distinct problems: detection of named entities, and classification of the named entities by the ontological type (i.e. concept). The first phase is akin to *segmentation* (which falls in the same category as other shallow parsing tasks e.g., *chunking* in the NLP community [97]). More generally, named entities are defined to be contiguous *non-nested* spans of tokens, so that 'President Barack Obama' is a single named entity, disregarding the fact that inside this name, the substring 'Barack Obama' is itself a valid named entity. This can sometimes lead to interesting problems for evaluating IE, especially at scale. Whether 'Barack Obama' or even 'Obama' would be tagged as correct instances of detection, as opposed to the more complete and unambiguous 'President Barack Obama' depends on the application and the way in which the evaluation is set up.

The second phase would require assigning a concept from T to the detected named entity. Mis-typed named entities would lead to errors in evaluation, as we describe towards the end of the chapter. Intuitively, we do not want to tag 'Tom Cruise' as a location rather than a person.

In practice, the situation can become rapidly more complex, since, in the definition of T above, there is no reason to restrict it to *just* a set of types. In fact, in a more complex task, such as event extraction, it is more reasonable to allow T to represent the *ontology* itself. Some event ontologies, such as CAMEO [180], can be very hierarchical and contain fine-grained classes like 'reject accusation', 'deny responsibility', 'express accord' and 'appeal for diplomatic cooperation'.

With such hierarchical ontologies, one would ideally want to tag a detected mention with the most fine-grained type, which is more difficult than distinguishing between coarse-grained concepts like Location and Person. For example, one could imagine an ontology that has fine-grained types such as 'Politician', 'Businessman', 'Celebrity', many of which would be linked to a super-type such as 'Person'. Given such an ontology, would it be incorrect to tag 'Tom Cruise' with type 'Businessman' or would 'Celebrity' be the only correct answer? How do we penalize a system which tags 'Tom Cruise' as 'Person' vs. a more sophisticated system that tags the mention as 'Celebrity'? In general, evaluating the outputs of NER with respect to such an ontology is itself a complex issue, and in practice, depends on the application. We do not consider the issue in detail here. In the rest of the section on NER, we restrict our attention to the most popular version of the problem, which assumes a single non-hierarchical set T of terms as the ontology. Later, we briefly discuss an exception to the availability of an ontology (Open IE [10]).

Before describing the classes of techniques that are generally used for NER, we note that the question of how to precisely *define* a named entity is important philosophically, but is almost always evident from the application context. A functional definition is that a named entity is simply a span of text that can be typed according to one or more classes in a pre-specified ontology. The last word on whether a named entity is correct or not mechanistically depends on the gold-standard, which can itself occasionally contain missing and wrong entries, both due to human error or because of annotator disagreements.

2.3.1.1 Supervised Methods

The current dominant technique for addressing the Named Entity Recognition problem is supervised machine learning. Such techniques have classically included Hidden Markov Models, Support Vector Machines (SVMs) and Conditional Random Fields (CRFs) [78, 100, 191]. Although CRFs have emerged as the popular technique due to superior performance, in general, all of these machine learning methods can be used to construct a system that is trained using a large annotated corpus. In essence, such a system creates disambiguation rules based on discriminative features. An obvious baseline statistical learning method that is often proposed consists of tagging words of a test corpus when they are annotated as entities in the training corpus. The performance of the baseline system depends on the *vocabulary transfer*, which is the proportion of words, without repetitions, appearing in both training and testing corpus. As reported in a review, in a vocabulary transfer study that was conducted on the MUC-6 training data, it was found that vocabulary transfer accounted for 21% of performance, with about 42% accounting for locations, 17% for organizations and 13% for person names [134]. What these numbers illustrate is that achieving *generalization* is an important requirement, but that considerable variance exists across common ontological types. Vocabulary transfer is also a good signal of the recall (number of entities identified over the total number of entities) of the baseline system but (on the flip-side) may be more pessimistic than it appears since some entities are more frequent than others.

2.3.1.2 Semi-supervised Methods

Unlike supervised learning, semi-supervised learning (sometimes also known as *weakly supervised*) is designed to significantly reduce labeling effort without removing the human in the loop altogether (unlike *unsupervised learning*, which is covered subsequently) [120, 149]. An example of an influential semi-supervised NER technique is 'bootstrapping', which generally requires a set of seeds for initiating the learning process. For example, a system aimed at recognizing case citations in legal documents might ask the user to provide a small number of example citations. Then the system searches for sentences that contain these names and tries to identify some contextual clues (e.g., surrounding words, or their word embedding representations) common to the provided examples. Subsequently, the system tries to find other instances of citations that appear in similar contexts. This learning process is reapplied to the newly found examples, to discover more new relevant contexts. By iterating and repeating this process, a large number of case citations and contexts will eventually be discovered. Although supervised learning continues to be state-of-the-art, some experiments in semi-supervised NER have yielded performance rivaling baseline supervised approaches. Recently, these methods have become even more important in the context of extracting named entities from Twitter with minimal supervision [107, 156]. Thus, specific techniques are worth considering. We provide a brief review of some influential methods below

Mutual bootstrapping Mutual bootstrapping, first introduced in [155] as multi-level bootstrapping, consists of *growing* a set of entities and contexts by starting with a handful of seed entity examples of a given type (e.g., Sharon Stone and Sylvester Stallone are entities of type Actor) and *accumulating* the patterns found around these seeds in a sufficiently large corpus. Contexts, which are like linguistic patterns, (e.g., starred in X, is the star of X) are also ranked and used to find new examples to achieve some semblance of generalization. The authors of the original paper note that performance can rapidly deteriorate as noise starts to creep into either the entity list or pattern list. Although the overall empirical results were mixed, the idea behind mutual bootstrapping proved to be highly influential and has been considered as a prominent approach since, with multiple proposed variants. For example, one variant is to use syntactic relations (subject-object pairs) to describe context around the entities, rather than simple linguistic patterns. Furthermore, rather than relying on human-generated seeds, the process could be automated by relying on existing NER system outputs instead. In yet another variant, distributional similarity has been leveraged to generate synonyms or words which are members of the same semantic class [34], which allows for more robust *pattern generalization* (which, broadly speaking, is what much of NLP is about [109]). For instance, for the pattern X holds his meeting on a Monday, synonyms for Monday would be

{Tuesday, Wednesday, Tue., …}, thereby enabling the induction of more novel and generalizable patterns. Several authors have demonstrated that, by applying this technique to large corpora (hundreds of millions of webpages) and starting only from a seed of 10 examples facts, it is possible to generate one million facts with a precision of about 88%, an impressive performance metric.

How should one go about selecting the unlabeled data on which such mutual bootstrapping methods (or their variants) can be applied? One way is to rely not just on large arbitrary collection of documents, but to select documents using information retrieval-like relevance measures [160]. Furthermore, selection of specific contexts that are rich in proper names and coreferences bring the best results in their experiments. In general, the data selection problem is still not completely solved, especially given the almost limitless amounts of data (on any domain) now available on the Web. This problem has emerged into its own, and goes by various names, including intelligent crawling and domain discovery. For a treatment on the subject, see [133].

2.3.1.3 Unsupervised Methods

Clustering is the quintessential unsupervised machine learning approach [79]. Clustering of entities can rely on the similarity of context, lexical resources (e.g., WordNet), and even lexical patterns and statistics computed on a sufficiently large. Some specific approaches are described below.

As one form of unsupervised clustering, [4] studies the problem of labeling an input word with an appropriate named entity type from WordNet, the primary approach being to assign a *topic signature* to each WordNet synset by merely listing words that frequently co-occur with it in a large corpus. Then, given an input word in a given document, the word context (words appearing in a fixed-size window around the input word) is compared to type signatures and classified under the most similar one.

In a similar vein, a method similar to the identification of hyponyms/hypernyms described originally in [74] was proposed to identify potential hypernyms of *sequences* of *capitalized* words appearing in a document. Other variants have been proposed as well.

The observation that named entities often appear together in several news articles, whereas common nouns do not, has also been leveraged as ‘background’ knowledge for unsupervised NER model building. For example, a strong correlation was found between being a named entity and appearing contemporaneously in multiple news sources, which has allowed the identification of rare named entities in an unsupervised manner and can also be effectively combined with other NER methods.

The authors in [59] proposed and Information Retrieval (PMI-IR) as a feature to assess whether a named entity can be classified under a given type. PMI-IR was originally designed to measure the dependence between two expressions using web queries. A high PMI-IR meant that expressions tended to co-occur. The technique was leveraged for unsupervised NER by creating features for each candidate entity (e.g., Sharon Stone) and a large number of automatically generated discriminator phrases like ‘is a movie star’, ‘starred in’, ‘won the Golden Globe’ etc.

2.3.1.4 Features

Even the brief descriptions of approaches listed above for supervised, unsupervised and semi-supervised NER show that *features* are extremely important for good performance, and more often than not, can prove to be decisive factors in the quality of a system. Furthermore, in domain-specific KGC systems, whether for biomedicine, chemistry or even space, features can play an even more important role than usual [158, 162].

Features are characteristic attributes of words or other (such as segmented noun phrases) that are especially important for robust and good performance of machine learning classifiers. As an example, consider a quantitative feature that counts the number of characters in a word. Features represent abstract properties of a unit that assist in the generalization capabilities of machine learning capabilities. Boolean features return either true or false (e.g., whether a word is capitalized or not), while quantitative features return numbers, whether real-valued or discrete. Features can also be *nominal* or *categorical*[1] e.g., an ‘identity-lowercase’ feature would simply return the lower-cased version of the word as its output.

Although features have come to be associated closely with machine learning, expert and rule systems also make use of features. We may choose to institute a rule in our NER that says that if (1) the capitalization feature returns true, and (2) the nominal feature ‘inc’ or ‘corp’ (and other variants that we decide a priori) is detected, then the ‘slot’ should be output by the IE system with type Organization. Although rules work well for regular sentences and constrained grammars and styles, real systems tend to be much more complex and rules have to be *induced* using learning techniques or have to be superseded (as state-of-the-work often does) by machine learning-based sequence-labeling using cutting-edge deep neural networks like Recurrent Neural Networks (RNNs), which use units such as Long-Short Term Memory (LSTMs) units for advanced state-of-the-art sequence labeling, or more classic Conditional Random Fields (CRFs).

Features most often used for NER can be categorized along three different axes, as described in the survey by [121]: *word-level*, *list lookup*, and *document and corpus*. We describe these below, followed by notes on some recent advancements in *representation learning* that have fundamentally influenced feature engineering [118].

[1]One could also imagine *ordinal* features, which would be helpful in tasks such as recommendation, but these tend to be less common in IE.

Word-level features Word-level features tend to describe the ‘’ of a word, including such aspects as word case, punctuation, special characters and numerical value. It is especially important not to underestimate the usefulness of digits when defining word features. Digits can be used to express such information as dates, percentages and intervals, but each of these tend to be expressed using specific patterns. For example, a pattern such as ‘aa?’ where ‘?’ expresses a placeholder for a sequence of consecutive numerical characters, could be used to express e.g., a flight number (for American Airlines). Similarly, one can define patterns such that two-digit and four-digit numbers can stand for years, one and two digits may stand for a day or a month and so on [188].

Morphological features, heavily featured in any sufficiently robust NLP pipeline, including for tasks extending beyond (or preceding) NER, relate to elements such as words affixes and roots. Words ending in ‘ist’ (physicist, radiologist) could be used, with some probability, to detect Person entities that are professionals. Words ending in affixes like ‘ish’ could indicate nationalities, or even languages e.g., Danish [22]. However, one must be careful, since there are exceptions. For example, it would not be correct to tag the word ‘apologist’ as a Person entity with a professional background. There is an obvious multi-lingual element to these features as well [13]. For these features to provide value, they must be combined with other features in sufficiently robust learning algorithms. Given a large background corpus, statistical methods can also be used to discover relevant morphological features that may have been overlooked by a feature engineer.

As the examples and feature categories above suggest, features can also be extracted by applying functions over words. Collins and Singer [46] provides an early example where the authors construct a feature function that operates by isolating and concatenating non-alphabetical characters from an input word. For example, given the word ‘T.J.Max’, the output of the feature function would be ‘..’. Many candidates like this function now exist in the hundreds of NLP papers describing both statistical and rule-based systems for a variety of problems, including NLP. One of the more popular and recurring features is character n-grams, introduced by [138], although variants of the idea had been around for a while, especially in the information retrieval (IR) community. This feature is important enough that we describe the procedure for extracting it below. It is also slightly unusual in that the output is not a single value but a set of values. In popular variants, the set may also be a multi-set or a ‘bag’ i.e. it may contain duplicate elements.

Character n-gram Feature Function Let us assume a character n-gram feature function, where n ($\geq$1) is known in advance, and a special character #. Given a character sequence (which could be a single word, but can also be a multi-word sequence; the only general constraint is that the input should be a sequence and that the sequence should not include the special character) $[c_1, \ldots, c_m]$ containing $m \geq 1$ characters, the function would first pre-pend and append n$-$1 #’s to the sequence, thereby creating a new sequence of length $m + 2(n - 1)$. Next, a window of size n is slid over the sequence from beginning to end, and the sub-sequence contained within this window is placed inside the output set.

As an example, suppose we wanted to extract 3-grams (called tri-grams) given the chunk ‘President Obama’. First, we would append and pre-pend n−1 #’s (##) to the sequence, yielding the new sequence (represented as a string for convenience) ‘##President Obama##’. Now, a window of size 3 is slid over the entire sequence to yield the set {##P, #Pr, Pre,. . . ,ama,ma#,a##}.

Despite its simplicity, tri-grams and bi-grams have some advantages that make them robust to artifacts like spelling errors. Also, characters at the beginning and end of the chunk are overweighted compared to characters in the middle. This is in recognition of the fact that beginnings and endings tend to be important when comparing named entities. Character n-grams are also complementary to another line of important techniques proposed in the information retrieval community decades ago for robust comparison of ‘bags of words’, namely token-based and set-based similarity measures like cosine similarity on tf-idf vectors.

Pattern features were introduced by [45] and then used by others [44]. The idea is to map words onto a small set of patterns over character types. The goal is to achieve robustness and generalization by performing such a mapping. For instance, a pattern feature might map all uppercase letters to ‘A’, all lowercase letters to ‘a’, and all punctuation to ‘.’. With such a representation, seemingly different words begin to look similar e.g., ‘ABC Corp’ and ‘NBC News’ would both map to the same pattern using just the simple rules described above. It is possible to have pattern features that are much more sophisticated, and one could also induce relevant type-customized patterns from a background corpus of named entities. Such sophistication can be a double-edged sword in its own way. For example, if the background corpus is too general, performance of such pattern features may not be well suited or applicable to more domain-specific problems. On the other hand, if significant effort is expended to develop such corpora or patterns for domain-specific problems, then it may not be easy to extend to new domains. Robustness should also be given priority; small changes in inputs or background corpora should not strongly affect results on unknown or unseen (but still same-domain) datasets.

List-lookup Features In knowledge graph construction systems that often rely on background knowledge, such as knowledge derived from public sources like Wikipedia, DBpedia and YAGO [8, 167], list lookup features are extremely important. In traditional work, terminology can be varied, with *gazetteer*, *lexicon* and *dictionary* all used interchangeably with *list*. Lists are useful not just for named entity detection (the probability that Buenos Aires is a named entity rather than Buenos and Aires separately, increases significantly when we observe a Buenos Aires in an external lexicon), but more importantly, for entity typing. In the case of Buenos Aires, it is unlikely we will ever observe the named entity in a context other than as having type ‘Location’ or ‘City’ (if finer-grained typing is supported). In other situations, there can be considerable ambiguity. In earlier examples, we described how locations can sometimes be confused with geopolitical entities and vice versa. Because of word polysemy [151], additional problems arise. In the case of domain-specific KGC, especially for non-English scenarios, good dictionaries may not even exist. It is also important to realize a fundamental limitation of knowledge bases like Wikipedia, which generally contain pages on ‘well-known’ entities. In many domains, it is the long tail of less well-known entities, relations and events that is of interest to analysts and users.

Sometimes, however, lists exist in ‘plain sight’. Common nouns listed in a dictionary are useful, for instance, in the disambiguation of capitalized words in ambiguous positions (e.g., sentence beginning). Mikheev et al. [117] reports that from 2677 words in ambiguous position in a given corpus, a general dictionary lookup allows identifying 1841 common nouns out of 1851 (99.4%) while only discarding 171 named entities out of 826 (20.7%). In other words, 20.7% of named entities are ambiguous with common nouns, in that corpus.

Many authors propose to recognize organizations by identifying words that are frequently used in their names, such as ‘Inc’, ‘Corp’, ‘Associates’ or even ‘Telecom’. Most approaches implicitly require candidate words to exactly match at least one element of a pre-existing list. However, we may want to allow some flexibility in the match conditions. We describe some possibilities below.

First, words can be stemmed (stripping off both inflectional and derivational suffixes) or lemmatized (normalizing for inflections only) before they are matched [43]. For instance, if a list of cue words contains ‘subsidiary’, the inflected form ‘subsidiaries’ will be considered as a successful match. For some languages [80], diacritics can be replaced by their canonical equivalent (e.g., naïve would be replaced by naive).

Second, candidate words can be fuzzily matched against a reference list using techniques like thresholded edit-distance [171] or Jaro-Winkler [44]. This allows capturing small lexical variations in words that are not necessarily derivational or inflectional. For instance, John could match Johnny because the edit-distance between the two words is sufficiently small. In fact, Jaro-Winkler’s metric was specifically designed to match proper names following the observation that the first letters tend to be correct while name ending often varies. Other string similarities exist for specific matching tasks.

Third, phonetic algorithms like Soundex or Metaphone can be used to match against a reference list [146], since such algorithms normalizes candidate words to a phonetic code such that names which sound very similar map to the same code e.g., Jon and John would map to the same code. Soundex, which is the oldest and best known of the phonetic algorithms, produces a code which is a combination of the first letter of a word plus a three digit code that represents its phonetic sound.

Document and Corpus Features In the most general case, document features are defined over both document *content* and *structure*. This section describes some features that go beyond simple single and multi-word expression features to include meta-information and statistics about documents and corpora.

In an early work on document-centric features, several authors extract features from documents simply by identifying words that appear *both* in uppercased and lowercased form in a single document [117, 152, 169]. Those words are hypothesized to be common nouns appearing dually in ambiguous (e.g., the beginning of a sentence) and unambiguous positions.

Another feature, which is recognizing multiple occurrences of a unique entity in a document, dates back to research in the field that started in the early 1990s [115]. Determining multiple aliases[2] of an entity is a variant of the famous coreference resolution problem [122], which is still not solved. In early research, deriving features from coreference is mainly done by exploiting the *context* of every occurrence (e.g., Obama signed a treaty, Obama said that taxes will not be raised, Obama declared a truce. . .). Deriving features from aliases is mainly done by leveraging the union of alias words (Sir, Elton, E., John).

Finding coreferences and aliases in a text can be reduced to the same problem of finding *all* occurrences of an entity in a document, a complex endeavor. In a well-known domain-specific example, for instance, the authors in [64] use 31 heuristic rules to match multiple occurrences of company names. As an example of a heuristic, two multi-word expressions match if one is the initial subsequence of the other. Cross-document coreference resolution is even more complicated, on which research continues to this date. Both supervised and unsupervised approaches have been proposed and compared [103]. One proposal is to use word-level features engineered to handle equivalences (e.g., dr. is equivalent to doctor), with relational features encoding the relative order of tokens between two occurrences.

Unfortunately, word-level features are insufficient for complex problems. A , for instance, *denotates* a different concept than the literal denotation of a word (e.g., does Boston stand for the city of Boston or the Boston Red Sox?). The issue boils down to one of semantic tagging in the sentence [143].

Beyond document content, metadata can also be directly used as features, although the usage does tend to be domain-specific. For example, one could use email headers as good indicator of person names. Many news articles often start with a location name. Sometimes the purpose is to calibrate probabilities e.g., document URLs can be used to bias probabilities of entities. Again, the knowledge can be highly domain-specific. An interesting fact that has been noted is that names (e.g., bird names) have high probability to be a *project name* if the URL is from a computer science department domain than not.

Advances in Feature Engineering: Word Embeddings The preceding discussions highlighted the importance of feature engineering. In prior work, the utility of the features was central in determining the actual effectiveness of a machine learning algorithm for information extraction. In fairly recent work however, dating to the early part of this decade, word embeddings have emerged as an excellent way to mitigate feature engineering effort [118]. The basic idea behind most word embedding algorithms is to map each word in the corpus to a low-dimensional (in comparison to the dimensionalities of other text featurization algorithms like tf-idf), continuous (i.e., real-valued) *vectors*. The dimensions tend to lie between 20 and 100, depending on the size of the corpus. The embedding dimensionality is

[2]Aliases of an entity are the various ways the entity is written in a document e.g., Sir Elton John, E. John.

a hyperparameter and is accepted as given by the embedding algorithm. Generally, these embeddings work in a completely unsupervised fashion i.e. they do not require labeled data, although various complicated variants are also able to take labels into account when generating the embeddings. In modern work, a neural network (which is usually not deep unlike, say, a convolutional neural network) like a *skip-gram* model or a *continuous bag of words* model is used for the actual optimization.

The intuition is as follows. Imagine that we are *globally* trying to derive vectors for all the words such that, for a given word (e.g., 'cat'), either its vector is predictive of the vectors of its context words (in this case, the words that fall within a short distance of cat), or the context word vectors are together predictive of the given word.[3] However, when optimizing, the 'window' that defines the context is slid over all words in the corpus, so even though the context considered is local, the smooth sliding of the window ensures that influence is gradually percolating, leading to vectors that are hard to interpret but that have some remarkable intuitive properties.

These intuitive properties were borne out in some of the early (in the modern era[4]) work on word embeddings. The best known of these, and still extremely popular, is the *word2vec* algorithm. Word2vec allows optimization using either skip-gram or CBOW. Empirically, skip-gram has been found to be slightly better in actual problem settings. When using word2vec to embed a 'common' corpus like Wikipedia or the Google news corpus, it was observed that the vectors for words falling in the same (or similar) semantic class e.g., dog, cat and horse, fell close together in the vector space. Even more interestingly, it was possible to compute analogies in the vector space. In a famous example, one could do a computation such as **King** $-$ **Man** $+$ **Woman** in the vector space, and if a sufficiently large and broad corpus (like Wikipedia) was available then the resultant vector for the expression above was found to be close to the vector for **Queen**. This result is obtained despite the fact that the algorithm never received any labels, or any knowledge about which words fall in the same semantic class etc. Other algorithms inspired by word2vec or obeying similar methodology include GloVe and bag-of-tricks [84, 140]. All of them are based on some notion of context based on word co-occurrence, although the specific optimization functions are different. The bag-of-tricks approach also uses *sub-word* information, which allows it to deal with misspellings and OOV (out of vocabulary) words. This can be useful, especially when robustness is an important issue.

Embeddings have become so popular that they have percolated to allied communities in the knowledge discovery community, including graph embeddings, network embeddings, document embeddings and even knowledge graph embeddings. We cover the latter in Chap. 4. Because of the popularity of the field (formally called

[3]What is used for the prediction and what is being predicted is the difference between CBOW optimization and skip-gram optimization.

[4]The concept of embedding things into low-dimensional vector spaces is not novel. Using neural networks like skip-gram and CBOW and showing that they uncover properties of words that we intuitively understand is a relatively novel phenomenon.

‘representation learning’), it has become an enormously influential and impactful research of area within machine learning. Most likely, we have not seen the last of it and work will continue to emerge in this area.

2.3.2 Relation Extraction

(RE) is the problem of detecting and classifying *relationships* between named entities (NEs) extracted from the text [9]. A relation usually denotes a well-defined (having a specific meaning) relationship between two or more NEs.We illustrate using the following examples (including the relation type with the arguments in parantheses):

1. Personal/Social: [Mary, Queen of Scots] was the royal cousin of [Elizabeth I]
2. Employment/Affiliation: [Albert Einstein] was one of the most distinguished faculty appointed for life at the [Institute for Advanced Study] in [Princeton, New Jersey].
3. Physical[5]: [Josephine] moved the [couch] from the left corner of the room to the right corner, so that it was next to her [aunt’s portrait].
4. Geographical: [India] and [Nepal] are neighboring countries.

As with NER, we note that the set of relationship types that are within scope for the RE system is specified by a pre-defined ontology, although much of the existing work on relation extraction has tended to rely on a few ontologies for evaluation, such as the highly influential ontology from the Automatic Content Extraction (ACE) program [50]. ACE focuses on binary (relations between two entities), rather than arbitrary n-ary, relations. The two entities involved are generally referred to as *arguments*. The ACE ontology defines a set of major relation types and their sub-types, examples of major types including physical (e.g. an entity is physically near another entity), personal/social (e.g. a person is a family member of another person), and employment/affiliation (e.g. a person is employed by an organization) types. ACE also makes a distinction between relation extraction and relation *mention* extraction. The former refers to identifying the semantic relation between a pair of entities based on all the evidence we can gather from the corpus, whereas the latter refers to identifying individual mentions of entity relations. This is an interesting deviation from how NER systems have evolved, which (even today) focus primarily on the extraction of mentions, leaving it to modules like (both within-document and cross-document) coreference resolution and entity resolution (covered in Chap. 3) to cluster mentions into the same underlying entity. In practice, however, because

[5]This particular example shows how a seemingly simple sentence in English can prove enormously difficult to capture in a structured semantic form. *Movement* in this context is a quarternary relation (*who* moved *what* from *where* to *where*), but the sentence also expresses a simpler ‘next-to’ relation.

corpus-level relation extraction largely relies on accurate mention-level relation extraction, the latter is of primary interest in any discussion on relation extraction.

The examples also illustrate how open-ended the RE problem can be in terms of arguments, relation types and granularities. Even in the simple *geographical* example sentence, one can see that multiple sub-types of relations can exist (neighbor-of, has-continent, located-in-country etc.) and each such sub-type can be expressed in different ways (we could have equivalently written the sentence as 'India and Nepal share borders') rendering the ambiguity problem in IE particularly challenging for even supervised systems.

For these reasons, despite its obvious importance to domain-specific knowledge graph construction, relation extraction systems do not currently enjoy the same levels of performance as state-of-the-art NER systems [98]. Generally, the more complex the definition of an extraction, the worse its corresponding extractor performs. For example, event extraction, which we cover subsequently, performs even worse than relation extraction, with even state-of-the-art systems far from the 70% F-Measure mark (with even worse performance in more novel domains than the ones where the system might have been trained) that is considered feasible for large-scale adoption. Much research is still left to be done for these extraction problem domains.

In fact, successful RE requires detecting the *argument* mentions (e.g., Barack Obama, United States), with the entity types chaining these mentions to the ontological types (e.g., Politician, Country) their respective entities, and the type of relation (e.g., PresidentOf) that holds between these arguments. Relation extraction faces several challenges, several of which are shared by NER, though not to the same extent. First, RE is much more dependent on the domain, and the language, then NER. Supervised machine learning techniques applied to RE face the usual difficulty of a lack of sufficient training data. Another interesting problem is that the notion of a relation is inherently ambiguous, which means that labeling itself can be a problematic endeavor, reflected in high inter-annotator disagreements. Extending binary RE techniques to RE involving higher arity is also problematic, as we describe in the section on Event Extraction (EE). Even in the simple example above, we can intuitively see that detecting or classifying Barack Obama as President of the United States is much easier than detecting that Barack Obama was the President of the United States from 2009 to 2016. Quickly, the problem can become intractable, or the noise makes results unusable.

Many of the earlier techniques that we mentioned for NER, including supervised, semi-supervised and unsupervised learning, also apply to RE. Examples of systems that are now considered relatively classic include DIPRE, Snowball, KnowItAll and TextRunner [2, 32, 59, 187], all of which are semi-supervised rather than fully supervised or unsupervised. The problem is modeled differently however. Assuming binary relations, one way to model RE is to train either a binary (per relation) or multi-class classifier for pairs of extracted NEs. However, if one does this for every pair of entities extracted from the document, the complexity quickly becomes quadratic, and performance declines sharply. Thus, heuristics often have to be used to impose constraints. For example, one might only consider applying such classifiers to pairs of NEs extracted within a given span of text, or even within the same sentence. Recently, deep learning methods have also been extensively applied to RE [98].

Similar to NER, feature engineering has also been an important issue, and word-level, semantic and kernel features have all been proposed over the years. Kernel methods were especially popular in the earlier part of the last decade, an influential work being [189]. Feature classes specific to relation extraction have also been proposed. In general, such classes become necessary for 'higher-order' extractions like relations and events. Also, as described earlier, word embeddings have had a major impact on all of NLP, and RE feature engineering is no different. A more exciting development has been the *joint modeling and extraction* of relations and entities [190], and more generally, the joint extraction of events and entities. We describe the intuition behind such joint models in more details in the next section. Also, as pointed out earlier, deep learning methods, which almost always include some form of representation learning, have also been applied to RE, making the feature engineering problem less of an ad-hoc effort [98].

Because of the difficulty of the RE problem, and its recency, performance is lower across the board compared to NER; furthermore, performance tends to decline much more sharply as supervision levels are lowered compared to NER. More details on RE approaches and evaluation, including fairly comprehensive surveys and methodology reviews, may be found in [19, 98] and [9]. As the authors in that paper describe, the field is still relatively new compared to NER and much work is left to be done, especially in multi-lingual RE, n-ary RE and improvements in state-of-the-art methods.

2.3.3 Event Extraction

(EE) refers to the task of identifying events in free text and deriving detailed and structured information about them, ideally identifying who did what to whom, when, where, through what methods (instruments), and why [142]. Event extraction involves extraction of several entities and relationships between those entities. For instance, in an example taken from [142], extracting terrorist attack events from the text fragment 'Masked gunmen armed with assault rifles and grenades attacked a wedding party in mainly Kurdish southeast Turkey, killing at least 44 people.' involves identification of perpetrators (masked gunmen), victims (people), number of killed/injured (at least 44), weapons and means used (rifles and grenades), and location (southeast Turkey). Just like relation extraction, the problem can be very domain-specific. For example, in the mergers and acquisition domain, an event might be a merger that has just happened, which would require extracting the companies undergoing the merger (usually asymmetrically), the underwriter, the dates, specific merger terms, attorneys involved etc. EE is considered to be the hardest of RE, NER and EE.

For geopolitical style events, EE has been mainly studied using the ACE ontology [50], though other alternatives, such as CAMEO and ICEWS also exist [180]. For the biomedical domain, the BioNLP shared tasks are popular [91]. Because intense research continues to be conducted in EE, it is much too early to say which techniques are established and will stand the test of time. Some strands have started to emerge, however. For example, to reduce task complexity, early work tended to employ a sequence of classifiers that first extracted event *triggers*, then determined the trigger arguments [3, 28]. With the advent of deep neural networks, Convolutional Neural Networks (CNNs) have been employed as the pipeline classifiers [130]. Regardless, pipeline approaches suffer from error propagation via cascading, and *joint extraction* methods have emerged as state-of-the-art as a result [102]. As the name suggests, joint IE approaches tend to extract event triggers and arguments *together*, using methods such as structured perceptron [102], and dependency parsing algorithms [114].

The intuition behind joint IE is worth considering, especially considering the systems-level nature of knowledge graph construction. One can think about it as follows. Events and entities are *closely related*; entities are often actors or participants in events and events without entities are uncommon. The interpretation of events and entities is highly contextually dependent. Existing work in information extraction typically models events separately from entities, and performs inference at the sentence level, ignoring the rest of the document. An alternate approach that has recently come into vogue is to model the dependencies among variables of events, entities, and their relations, and to perform joint inference of these variables across a document. In essence, the learning problem is decomposed into three tractable subproblems: learning within-event structures, learning event-event relations, and learning for entity extraction. Probabilistic models are learned for all of these subproblems, with a joint inference framework integrating the learned models into a single model to jointly extract events and entities across a document.

The experimental results have been quite impressive, achieving state-of-the-art performance on EE typed according to the ACE ontology. Even more importantly, it was found that the benefits would often be mutual i.e. even a well-studied task like entity extraction benefited from better performance when done in the context of a joint inference framework. This is in line with the intuition stated earlier that there are close semantic connections between event triggers and arguments, and it is best to model such connections explicitly in the extraction framework itself.

One limitation of joint IE approaches is that they can suffers from complexity issues, like joint or collective approaches in general. To combat this problem, some approaches tend to rely on heuristic search to aggressively shrink the search space. One sophisticated exception is [154], which uses dual decomposition for joint inference with runtime guarantees. Other approaches proposing to do joint IE without enormous complexity burdens are continuing to slowly emerge from the research community.

Overall, the performance of EE is quite poor compared to NER. One avenue of research for improving EE performance is to exploit *document-level contexts*. Berant et al. [16] exploits event-event relations, e.g., causality, inhibition, which frequently occur in biological texts. It is clear that such relations are domain-specific. For more general texts, existing work tends to focus on exploiting temporal event relations [35, 54, 113]. For the ACE domain [50], there is work on utilizing event type co-occurrence patterns to propagation event classification decisions [81, 104]. Intuitively, co-occurrence is generally a useful feature (e.g., a DIE event tends to co-occur with ATTACK events but interestingly also, TRANSPORT events). In recent years, more general approaches have been proposed in this vein that can handle broad-domain event relations (e.g., causal and temporal) through the design of appropriate features. Similar to other extraction problems, ontological constraints can also be leveraged. For example, an entity mention of type PER can only fill roles that can be played by a person. The empirical utility of ontological constraints for such tasks is only starting to be studied in detail, and we will likely learn a lot more about the benefits and limitations of such constraints in the periods to come.

2.3.4 Web IE

The Web has emerged as the single biggest source of data for many domains, including reviews, e-commerce, academic literature and even investigative domains like securities fraud and human trafficking. Given a domain, one problem is to find and crawl relevant pages from the Web. This problem is known as domain discovery, and although recent research on it has made much progress (using techniques like reinforcement learning and page classification, for example), it continues to be an interesting and difficult area of study.[6] However, even given such a corpus, constructing a domain-specific KG involves extracting important pieces of information from the webpages.

Web IE [36], which covers this problem broadly, has an extensive history, with the vast majority of influential papers being published about 10–15 years within the initial growth of the Web. The dominant technique is a ‘wrapper’ [99], which was originally defined as a component in a Web information integration system aimed at providing a single uniform query interface to access multiple information sources. In the case where the information source is a Web server, a wrapper must query the Web server to collect the resulting pages via HTTP protocols, perform information extraction to extract the contents in the HTML documents, and finally integrate with other data sources. Due to historical reasons, the term ‘wrapper’ is now almost exclusively associated (in the IE community) with Web IE.

[6]Two reasons are the dynamic nature of the Web, but also the safeguards often put in place (such as registration requirements, and captchas) to avoid crawlers.

At a high level, wrapper induction (WI) is the process used to generate wrappers, usually using semi-automatic, rather than manual or fully automatic, methods. Broadly speaking, a wrapper performs a pattern matching procedure relying on a set of extraction rules. Tailoring a WI system to a new task can be challenging, depending on the text type, domain, and scenario. To maximize reusability and minimize maintenance cost, designing a trainable WI system has been an important topic in Web knowledge discovery and domain-specific search. Unlike NLP-centric IE, covered before, Web IE processes online documents that are semi-structured and is consumed by a server-side application program that is attempting to ingest the information in the documents into some kind of a database that is ultimately accessed as a knowledge graph. Unlike traditional IE, Web IE is often not able to leverage techniques such as lexicons and grammars to the same extent, since HTML is very different from natural language, and has to instead rely often on exploiting a mix of heterogeneous features, such as syntactic patterns, layout structures of template-based documents as well as more traditional text-based features.

Similar to NER systems, WI systems can be classed as supervised, semi-supervised and unsupervised [36], although sometimes the distinctions aren't completely clear.[7] Supervised WI systems take a set of web pages labeled with examples of the data to be extracted and output a wrapper. The user provides an initial set of labeled examples and the system (perhaps with the help of a GUI) may suggest additional pages for the user to label. The advantage of using a GUI is to empower general users, rather than programmers, to use the system and label additional data, which permits greater applicability. In this sense, supervised WI systems are different from supervised NER systems.

Regardless, labeling examples with precision has always been considered to be a difficult and arduous task in the broader AI community. Semi-supervised WI systems like OLERA and Thresher try to find a way around this problem by accepting a rough set of (instead of a complete and exact set of) examples from users for extraction rule generation [37, 77]. Systems like IEPAD do not require labeling [38], but instead push effort to the post-processing stage, when the user is asked to choose a target pattern and indicate the data to be extracted. All these systems are targeted for record-level extraction tasks. In the KG context, a record can be thought of as an entity of interest, along with attributes describing that entity. For example, a record describing a product would have the product identified as the central entity, and attributes like price or description would be attributes that constitute the non-ID 'columns' of the record. In this way, a KG can be incrementally constructed by collecting such record-level extractions over a corpus of webpages. However, since no extraction targets are originally specified for such systems, a GUI is still required for users to specify intuitive extraction targets after the learning phase.

[7]For example, if the system required a lot of model engineering but no training data, is it really 'unsupervised'?

Purely unsupervised Web IE systems do not use any labeled training examples and have no user interactions to generate a wrapper. The best examples are RoadRunner and ExAlg [6, 49], which were designed to solve page-level extraction tasks, while systems like DeLa and DEPTA are better-known for record-level extraction tasks. Unsupervised systems 'discover' the extraction target by segmenting the data that is used to generate the page or isolating non-tag texts in 'data-rich' regions of the page. The overall problem tends to be severely under-constrained, making it difficult e.g., several schemas may comply with the training pages leading to ambiguity as to which attributes are important and which ones are not pertinent to the domain. The choice of determining the right schema is left to users, meaning the system is not fully unsupervised after all. Similarly, if only some of the extractions are relevant, post-processing may be required for the user to select relevant data and name the extracted clusters appropriately. The general goal however is to ensure that supervision is minimal compared to semi-supervised and supervised WI systems. A downside of this automation is that the system may become overly dependent on the layouts of pages in the development and model engineering phases, and may not do well (or even crash) in test phases or when subjected to new domains. In other words, the price of high automation can sometimes end up being a lack of robustness and generalization.

Because unsupervised WI continues to be a difficult area of research, relatively few systems exist compared to supervised systems. Earlier, we described *RoadRunner* as a highly influential example, whose impact has continued to be felt many years after it was first proposed. RoadRunner considers the *site generation* process as encoding the original database content into strings of HTML code [49]. As a consequence, data extraction is considered as a *decoding* process. Therefore, generating a wrapper for a set of HTML pages boils down to the inference of a *grammar* for the HTML code. RoadRunner uses a matching technique to compare HTML pages of the same class and generates a wrapper based on their similarities and differences. The main idea is to start by comparing two pages, using the ACME (which stands for Align, Collapse under Mismatch, and Extract) technique, described in the original paper [49], to align the matched tokens and collapse for mismatched tokens. Since there can be several alignments, RoadRunner adopts *union-free regular expressions* to reduce the complexity of the process. The alignment result of the first two pages is compared to the third page in the page class, and the process continues.

Although the techniques details of RoadRunner can become complex, it is worth noting that, unlike other wrapper induction techniques that generated wrappers by examining labeled examples and has knowledge of the target schema, RoadRunner does not have prior knowledge about the organization of the pages. The technique is also quite efficient, since the authors proposed various mechanisms (such as union-free regular expressions) to ensure that the complexity does not exceed practical limits. In the original paper, RoadRunner was able to outperform supervised wrapper induction techniques like Wien and Stalker on an efficiency metric (CPU time) by orders of magnitude [49].

2.4 Evaluating IE Performance

Like any class of algorithms that has been rigorously researched and improved over decades, IE can also be evaluated on multiple metrics of interest (of its output) to a consumer. Many of these metrics quantify the difficulty of the problem via tradeoffs. Two metrics that we pay special attention to, and that will also play a role when we describe the evaluation of Entity Resolution in the next chapter are *recall* and *precision*, which were originally adopted from the Information Retrieval research community. These metrics, defined subsequently, can be respectively seen as measures of *completeness* and *correctness*.

Since the metrics will be computed on the output of an IE *with respect to* a gold-standard set of annotations, it is worthwhile asking what the outputs and gold-standard look like. Given a corpus of documents, each document *d* can be defined as a sequence of characters such that any (consecutive) *span* within the document may be identified using a *start offset* and an *end offset*. In this simple model, the gold-standard may be thought of as a set of triples, where each triple is of the form *(d, start offset, end offset)*. The IE output will also have this format. Let us define each element of the gold standard as a *slot*. Intuitively, the IE system ingests a corpus of documents, and outputs a set of candidate fillers for the slots. The goal is to evaluate the IE system's candidate slots against the reference slots in the gold standard. Although the NER systems that were covered fall very naturally in this category (with entity extractions filling slots), relation extractions and even event extractions (broken down into argument and trigger detection) can be defined in a similar way.

As a first step, let $numSlots$ denote the number of slots in the gold standard G (a set of slots), and let the IE output O be the set of candidate slot fillers. We define the set of *true positives* to be $O \cap G$ i.e. the slots in O that were correctly extracted. Let the set of *false negatives* be $G - (O \cap G)$ i.e. the slots in G that were never extracted by the IE system. Finally, we refer to the slots in O that do not occur in G as *false positives*.

Using these definitions, precision and recall are defined as follows:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.1}$$

$$recall = \frac{true\ positives}{numSlots} \tag{2.2}$$

This simple model of IE outputs and gold standard is appealing because it can easily be extended to other, more complex, IE problems. For example, in a particular kind of IE system (NER), the system must not only identify the slots but must also *type* the slots according to concepts from an ontology. Examples of concepts, as covered earlier, include ontological elements such as Person, Organization and Location. Per this notion, a slot in a gold standard would have the form *(d, start offset, end offset, type)*. For a more fine-grained picture of an IE system's performance, precision and recall are often measured for each slot type separately. The F-measure is used as a weighted harmonic mean of precision and recall, which is defined as follows:

$$F = \frac{2 * precision * recall}{precision + recall} \tag{2.3}$$

Precision, recall and F-measure are metrics that are used quite broadly within computer science and are largely influenced by developments in the Information Retrieval (IR) community. However, some metrics are IE-specific. One example is the slot error rate (SER) which characterizes the extent to which the IE system makes mistakes (as opposed to the other metrics defined above, which characterize either correctness or completeness of the IE system). To define the SER, we must first assume that slots in the IE output O are *aligned* (usually algorithmically) to the slots in the gold standard G. A simple, but very conservative, definition of alignment is that both the start and end offsets must be equal for two slots to be aligned. Note that if the IE problem involves entity types, such as in NER, two slots (one from O and one from G) can be aligned but not match, since the type in each slot could be different. For example, it is possible that an extraction 'United States' got typed as a location by the IE system, but is actually a geopolitical entity in the gold standard. We assume that a slot in the gold standard can be aligned with at most one slot in G. With this in mind, let us define the variable *#wrong* as the number of slots in G that are (1) aligned with some slot in O, (2) do not match the aligned slot in O. In a similar vein, let us define *#missing* as slots in G that are not aligned with any slot in O. With these variables in place, SER can be defined as:

$$SER = \frac{\#wrong + \#missings}{numSlots} \tag{2.4}$$

The intuition behind SER is relatively simple: how many of the slots in the gold standard G did the IE system either miss or get wrong? Similar to the other metrics, SER is always between 0.0 and 1.0, but unlike the other scores, a lower SER indicates a higher quality IE system.

Metrics other than SER, precision, recall and F-measure also exist for characterizing IE system quality. Most modern systems, however, tend to focus on these metrics.

2.5 Summary

Information Extraction (IE) is the first, and possibly most important step, in a domain-specific knowledge graph construction system, once preliminary steps such as domain discovery and dataset collection have been performed. Like most AI problems, IE is not a solved problem, though performance has continued to steadily improve over the years, including for semi-supervised and unsupervised IE. Modern advances, especially in event extraction, have been quite exciting, particularly due to deep neural networks, and more recently, generative adversarial networks (GANs). In the next several chapters, we cover downstream steps such as Entity Resolution and Knowledge Graph Completion that must consume the noisy outputs of IE systems.