

Chapter 3

Searching for Named Entities

In the previous chapter, we searched for the specific composer *Ludwig van Beethoven*. But what if we wanted to find sentences about *any* classical music composer, or even more generally, about any composer? So far our strategy has consisted of starting with a URI, finding possible alternative surface forms, and looking for sentences in which they occur. If we follow the same strategy in the case of *composers*, we can find the URI `dbr:Composer` in DBpedia and discover some of its surface forms (e.g., *music composer*, *musical composer*, and even *author*) through the predicate `dbo:wikiPageRedirects`.

Would a search in a corpus for sentences containing these surface forms be likely to lead us to information about composers? Perhaps it would, but it is certain that the recall performance of such approach will be quite low. For example, if we attempt the above strategy on the `BEETHOVENCORPUS` from the previous chapter (see Table 2.1 in Sect. 2.1), we find a single sentence (sentence no. 6) among the 10 sentences about composers, which explicitly uses the word *composer*.

An alternative approach, presented in this chapter, is to consider `COMPOSER` as an **entity type**, also referred to as **entity class**. An entity type, or entity class, represents a set of individuals, and we will develop **text mining** strategies for finding the individuals which belong to this class.

The first strategy is using a list of the individuals in the class. For example, we can gather a list of composers, such as *L.V. Beethoven*, *W.A. Mozart*, and *J.S. Bach*, and search for them in text. In NLP, such list is often referred to as a **gazetteer**.

The second strategy is to search for regularities in the way of expressing the individuals in the class. The type `COMPOSER` is likely not the best candidate for this strategy, although being a subclass of `PERSON`, we can expect the same regularity in a composer's name than in a person's name. Such regularity could be a sequence of two capitalized words (e.g., *[F]rank [Z]appa*), although we can imagine such regularity leading to many other entity types than `PERSON`, such as `CITY` (e.g., *[N]ew [Y]ork*) or `COUNTRY` (e.g., *[S]ri [L]anka*). Other entity types, such as `DATE` or `TIME`, are better candidates for this regularity detection strategy.

We will look at various entity types in this chapter, as well as the use of gazetteers and the detection of regularities through a very powerful tool, **regular expressions**. We will also continue with the experiment-based learning approach promoted in this book, by defining an experiment to test the precision and recall of regular expressions in search of DATE in a corpus.

3.1 Entity Types

COMPOSER is simply an example of an **entity class**, also called an **entity type**, a **semantic type**, or a **category**. SYMPHONY would be another example, as would be AUTHOR, COMPANY, and CITY. Entity types are at the core of knowledge organization, since they provide a structure for our understanding of the world.

And because entity types play a central role in our quest for information, we will often return to these notions, and even debate their definitions. For example, is *cell phone* an entity type? Not in the same way as COMPOSER is, yet we can talk about cell phones in general, and individual phones do exist under the broader umbrella of CELLPHONE. Although in our quest for information, we would rarely be interested in the specific phone Mr. X owns, but we might be interested in a particular Samsung or Nokia phone, just being put on the market. My purpose here is not to enter in a philosophical debate about ontological commitment or separation of classes and individuals. I simply want to bring awareness to the impact the difference between searching for entity types versus individuals can have on searching strategies and search results.

Compared to the previous chapter, in which we discussed generic and specific entities, notice how the current chapter introduces a different terminology, more commonly used in NLP and in the Semantic Web, of individuals and entity types. In the previous chapter, we used the term **specific entity** to refer to an individual and **generic entity** to refer to an entity type.

Interest in individuals and entity types is pervasive in the Semantic Web and NLP communities. The term **Named Entity Recognition**, abbreviated **NER**, refers to an important field of research within NLP aiming at recognizing named entities in corpus. We yet introduce another term: **named entity** which sense is closest to what we had called specific entity. In a strict sense, a named entity is an instance of an entity class, uniquely identified via a name. In this same strict sense, named entities are unique individuals. People, organizations, locations, and dates are all examples of things that are unique in our world. But a NER search might be interested in detecting other *important information* in a text, such as amounts of money or quantities. This extends the definition of named entity toward a less strict sense including individuals as well as other precise and important information.

As the reader, you might find this confusing to be introduced to many similar terms having partially overlapping meaning. It is confusing, but it is important to know about all these terms, since you are likely to encounter them in different books and research articles written over many years. The effort to define types of named

entities and recognize them in text goes back twenty years to *The Sixth Message Understanding Conference* during which an NER task was introduced. At that time, the named entities to be recognized were given one of three possible labels:

- ENAMEX: PERSON, ORGANIZATION, LOCATION
- TIMEX: DATE, TIME
- NUMEX: MONEY, PERCENTAGE, QUANTITY

Later, efforts were made to define more fine-grained lists of entity types, some examples being:

- PERSON: ACTOR, ARCHITECT, DOCTOR, POLITICIAN
- ORGANIZATION: AIRLINE, SPORTS_LEAGUE, GOVERNMENT_AGENCY, NEWS_AGENCY
- LOCATION: CITY, COUNTRY, ROAD, PARK
- PRODUCT: CAR, CAMERA, COMPUTER, GAME
- ART: FILM, PLAY, NEWSPAPER, MUSIC
- EVENT: ATTACK, ELECTION, SPORTS_EVENT, NATURAL_DISASTER
- BUILDING: AIRPORT, HOSPITAL, HOTEL, RESTAURANT
- OTHER: TIME, COLOR, EDUCATIONAL_DEGREE, BODY_PART, TV_CHANNEL, RELIGION, LANGUAGE, CURRENCY

Beyond these more generic entity type lists, efforts have also been made in individual domains to define entity types specific to them (e.g., GENE and PROTEIN in biology).

Once entity types are defined, how would we devise a text mining process to identify in a corpus sentences mentioning these types. A first strategy is to use lists of individuals belonging to these types, as we see next.

3.2 Gazetteers

One approach to finding named entities in text is to have lists of the individuals, often referred to as **gazetteers**. On the Web (e.g., in Wikipedia), we can find lists of just about anything imaginable: varieties of rice, car brands, romantic symphonies, countries, and so on. Let us take art museums as an example. The following query submitted to DBpedia SPARQL endpoint would provide a long list of hundreds of museums:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X where {
    ?X rdf:type dbr:Art_museum .
}
```

This list could easily become a gazetteer for an ARTMUSEUM entity type to be used for searches in text. Table 3.1 provides some examples. Each of the entities has a variety of surface forms, all of which could be used in a search.

Table 3.1 Gazetteer for ARTMUSEUM

No.	Museum label
1	Berkshire Museum
2	The Louvre
3	Museum of Modern Art, Antwerp
4	Hirshhorn Museum and Sculpture Garden
5	Museum of Fine Arts of Lyon
6	Kosova National Art Gallery
7	Art Gallery of Algoma
8	National Gallery of Canada
9	Museu Picasso

Similarly, using the following query into DBpedia SPARQL endpoint, we can find many composers classified under the *Viennese Composers* category in Yago.¹

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>
prefix yago: <http://dbpedia.org/class/yago/>
select ?X where {
    ?X rdf:type yago:VienneseComposers
}
```

Results of the query can be used to generate a gazetteer for the VIENNESECOMPOSER entity type, as in Table 3.2. This list would include many contemporaries of *Ludwig van Beethoven*, the individual of our earlier investigation.

Table 3.2 Gazetteer for VIENNESECOMPOSER

No.	Composer label
1	Wolfgang Amadeus Mozart
2	Franz Schubert
3	Johann Strauss I
4	Franz Lehar
5	Ludwig van Beethoven
6	Johannes Brahms
7	Joseph Haydn
8	Anton Webern
9	Alban Berg
10	Arnold Schoenberg

¹Yago is a large Semantic Web resource, described at <http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/>.

An unfortunate drawback to all lists is that they are likely to be incomplete and thus insufficient. Despite this, lists are important in the search for named entities, and are widely used, most often in combination with other strategies. In Chap. 13, as we revisit entity types as constraints for relation search, we will investigate voting strategies to combine gazetteers with other methods of entity type validation.

For now, let us explore a second strategy for finding mentions of an entity type in a corpus, using regular expressions.

3.3 Capturing Regularities in Named Entities: Regular Expressions

Certain entity types have fairly regular ways of expressing their individuals, which we can easily detect. The individuals for the entity type ARTMUSEUM, as shown in Table 3.1, do include a few words that appear repeatedly (museum, fine art, art gallery), but still, we would have to consider more data to better convince ourselves of regularity within that class. What about an entity type SYMPHONY, to continue on our musical theme, would that contain regularities?

Table 3.3 Examples of instances of the SYMPHONY type

No.	Symphonies
1	Symphony no. 8
2	Beethoven’s symphony no. 6
3	Symphony no.4, op. 47
4	Symphony no 5 in b flat
5	Symphony no. 1 in d minor
6	Schubert’s Symphony no. 4
7	Symphony “pathetique”
8	Symphony no. 3 in b-flat minor
9	Abel’s Symphony no. 5, op. 7

Consider a short list of individuals, belonging to the SYMPHONY type, as shown in Table 3.3. As you can see, there is still quite a lot of variation within these examples, but we can identify certain patterns emerging. In an attempt to concisely capture and represent the variations seen here, we turn to **regular expressions**.

Regular expressions provide a concise way of expressing particular sequences of text. Although they can seem intimidating at first, it is essential to master them for NLP, since they provide a very flexible and powerful tool for text search. Most programming languages define text search libraries allowing the use of regular expressions. The best way to become familiar with writing these expressions is

simply to practice. The process is comparable to learning a new language, practice is key.²

Below are examples of what regular expressions can capture in text:

- single character disjunction: `[tT]able` → table, Table
- range: `[A-Z]x` → Ax, Bx, Cx, Dx, ... Zx
- explicit disjunction: `[a-z|A-Z]` → all letters
- negation: `[^Ss]` → everything except uppercase and lowercase 's'
- previous character optional: `favou?r` → favor, favour
- repetition of 0 or more times: `argh*` → arghhhhhhh, arghh, arg
- repetition of 1 or more times: `x+` → x, xx, xxx
- wildcard on a single character: `ax.s` → axis, axes
- specification of number of characters: `x{2,4}` → xx, xxx, xxxx
- escape for special characters: `\(a` → (a

As you can see, the language provided by regular expressions includes five important aspects: repetition, range, disjunction, optionality, and negation. As we can combine these aspects in various ways and apply them on either single characters or groups of characters, there is an infinite number of text segments which can be captured with regular expressions. That explains how regular expressions capture the representation of very long lists within a single expression.

Granted, for the SYMPHONY type, it may be possible to list all the existing symphonies, but what about entity types such as DATE, PHONENUMBER, or EMAILADDRESS? Regular expressions allow for the concise representation of variations within these non-enumerable entity types. Table 3.4 shows examples of different regular expressions to recognize particular entities.

²There are some online regular expression testers, such as *Regexpal*, available at <http://regexpal.com/> which allow you to write regular expressions and use them to search in text. You can also use the regular expression matcher libraries within your favourite programming language to write and test the search capacity of regular expressions.

Table 3.4 Examples of regular expressions for specific entity types

RegEx	Examples
Abbreviated winter month, optional final period (Jan Feb March) . *	Jan. / Feb
Any year between 1000 and 3000 [12][0-9]{3}	1977 / 2015
Postal codes in Canada [A-Z][0-9][A-Z][0-9][A-Z][0-9]	H2X3W7 / Q1Z4W8
Avenue names (1st 2nd 3rd [4-20]th) (Avenue Ave. Ave)	3rd Avenue / 6th Ave.
Any street name [A-Z][a-z A-Z]* (Street St. St)	Wrench St. / Lock Street
North American phone numbers \ ([1-9]{3}\) [1-9]{3} (-*) [0-9]{4}	(456) 245-8877 / (123) 439 3398
Symphony name [Ss]ymphony [Nn]o. * *[1-9]	Symphony no.4 / symphony No 5

Notice how we require **repetition** for capturing sequences of 3 digits in a row for telephone numbers, **range** for capturing the notion of a digit (1–9), **disjunction** to provide alternatives for months, and **optionality** to allow street names to contain a period after St or not. Out of the five aspects of regular expressions, only **negation** is missing in Table 3.4.

Now, let us design an experiment to put regular expressions to the test and study their behavior in an entity type search.

3.4 Experiment — Finding DATE Instances in a Corpus

Let us now tackle the problem of finding dates. Dates are important in Information Extraction, since we often want to know *when* things happen. Moreover, they represent a type of entity for which the enumeration of instances would be much too lengthy to be practical. Because of this, DATE is an entity type where regular expressions should prove quite useful.

3.4.1 Gold Standard and Inter-Annotator Agreement

First, we need to establish a gold standard against which we can evaluate our algorithm through its phases of development. Table 3.5 provides such a possible gold standard.

Table 3.5 Examples of sentences containing (or not) a DATE

No.	Sentence	DATE instance?
1	He was born on the <i>8th of July 1987</i> .	Yes
2	Cancel the flight 2000 by Sept. 10th to avoid paying fees.	No
3	A date to remember is <i>August 10, 1999</i> .	Yes
4	I wrote “ <i>10/04/1999</i> ” on the form, next to my name.	Yes
5	Class 20–09, that was a nice course.	No
6	The wedding was in 2008, <i>on April 20th</i> exactly.	Yes
7	Flight Air Canada 1987, is 6th in line to depart.	No
8	The expiration date is <i>15/01/12</i> .	Yes
9	Your appointment is on <i>October 31, 2012</i> .	Yes
10	It is Dec. 15 already, hard to believe that 2011 is almost over.	Yes
11	<i>November 1986, on the 7th day</i> , the train departed.	Yes
12	There was 5000 mm of rain in 7 days.	No
13	He was in his 7th year in 1987.	No
14	The product code is 7777–09.	No
15	He arrived 31st in rank at the 2005 and 2008 runs.	No
16	The event happens <i>March 10–12 2015</i> .	Yes
17	The big day is <i>November 20th, 2017</i>	Yes

When creating a gold standard, it is tempting to include only target sentences. In our case, this would mean exclusively including sentences that contain dates. The problem with this approach is that it would not allow us to evaluate the number of false positives produced by our algorithm. After all, if all the sentences qualify for our search, then we have not given the algorithm the chance to falsely identify one as qualifying when it does not. And, as we saw in Chap. 2, Sect. 2.2, identifying false positives is an important part of the evaluation of an algorithm’s precision.

For this reason, we should intentionally include non-date sentences in our gold standard. To further ensure the precision of our algorithm, we should contrive these negative examples to differ only slightly from the positive ones. You can think of this as testing the algorithm’s precision by including sentences that are almost, but not quite, what we are looking for, and seeing whether it ‘takes the bait’.

In the last column of Table 3.5, you will notice that I have marked examples as either positive or negative. Do you agree with my decisions? We might agree on some, but probably not all examples. Since there are many possible interpretations of what qualifies as a positive or negative example of an entity, annotators (human judges) are rarely in full agreement until the task is very well defined. I will reiterate my classifications below (Judge 1) and invent another hypothetical set, which I will attribute to a fictive other judge (Judge 2).

Judge 1: {1, 3, 4, 6, 8, 9, 10, 11, 16, 17}

Judge 2: {1, 2, 3, 4, 6, 8, 9, 10, 11, 13, 15, 16, 17}

What we now have before us is a problem of **inter-annotator agreement**. Within NLP, we talk of inter-annotator agreement to represent the degree to which two human judges agree on a particular annotation task. In the current example, the annotation was to determine Yes/No for each sentence, as containing an instance of a DATE or not. Inter-annotator agreement is an important concept to understand, since it will come back again and again in the development of NLP evaluation sets.

Certain measures have been proposed for the specific purpose of determining levels of inter-annotator agreement. One such measure is **Cohen’s Kappa**. This measure requires a matrix similar to the contingency table we used earlier (see Sect. 2.2) for comparing an algorithm to a gold standard. Table 3.6 shows where we (*Judge 1* and *Judge 2*) agreed and disagreed in our earlier classifications of date and non-date sentences.

Table 3.6 Judge comparison for DATE search example

		Judge 2	
		correct	incorrect
Judge 1	correct	10	0
	incorrect	3	4

The Kappa measure also requires that we account for agreement that could occur by chance alone. In other words, what is the probability $Pr(e)$ that closing our eyes and marking Yes/No beside the various sentences would result in agreement? Once we know the value of $Pr(e)$, we can compare our actual agreement $Pr(a)$ against it. Both $Pr(a)$ and $Pr(e)$ are part of the Kappa measure, as shown in Eq. 3.1.

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \quad (3.1)$$

Since we only have two classes in this case (Yes/No), our probability $Pr(e)$ of agreement by chance is 50 %. If there were ten different classes, the probability of two judges agreeing would be reduced to 10 %. In cases of open set of annotations, when the number of classes is not set in advance, measuring such agreement becomes quite difficult.

Our actual agreement $Pr(a)$ is the number of times both judges said Yes (10 times) + the number of times both judges said No (4 times) divided by the total number of examples (17).

$$\begin{aligned} Pr(a) &= \frac{(10 + 4)}{17} = 0.824 \\ Pr(e) &= 0.5 \\ \kappa &= \frac{(0.824 - 0.5)}{1 - 0.5} = 0.65 \end{aligned}$$

So, does a κ of 0.65 mean we agree? How should we interpret this result? Although there is no universal standard for interpreting Kappa scores, the scale below is often used.

$\kappa < 0$ would indicate no agreement
 $0 > \kappa < 0.20$ slight agreement
 $0.21 > \kappa < 0.40$ fair agreement
 $0.41 > \kappa < 0.60$ moderate agreement
 $0.61 > \kappa < 0.80$ substantial agreement
 $0.81 > \kappa < 1.00$ almost perfect agreement

Based on this scale, we have attained substantial agreement. This is not a bad result, but we could do better. A less-than-perfect measure of inter-annotator agreement can be taken in two ways: either the task is not well defined or the task really is highly subjective. In our case, we could potentially reach a higher level of agreement if we refine the definition of our task. In order to accomplish this, we should ask ourselves certain questions: What does it mean to find dates? Are we only interested in finding full dates that we can map to a calendar day? Do we want year only mentions to be included? Do we want to include ranges of dates?

The answers to these questions will be determined by how we intend to apply our results. To clarify this intention, we should consider why we are extracting dates and what will we do with them once we have them. In our current experiment, we have not yet defined a particular application, but we can still try to be as specific as possible in defining our task. My criterion for answering Yes in Table 3.5 was that the sentence must contain a date specific enough that I can map it to a calendar day. Knowing this, do you now agree with my classifications? If so, what would our new Kappa be?

3.4.2 Baseline Algorithm: Simple DATE Regular Expression

For our first exploration, let us come up with two regular expression baselines, one generic (high recall) and one very specific (high precision), and define them. Although up to now we have mostly seen regular expressions written as one long string, when programming them we can take advantage of the fact that programming languages allow us to use variables to contain strings. This further allows us to define partial elements of the regular expressions and to later combine them in different ways. In the example below, I first define regular expressions for months, years, and days and then I show how to combine them using a string concatenation operator (+), to generate the two baselines:

```

Year      = [12] [0-9] {3}
Month     = (January|February|March|April|May|June|July|
            August|September|October|November|December)
DayNum    = ([1-9] | [1-2] [0-9] | 30 | 31)
Baseline 1 : Year
Baseline 2 : Month + " " + DayNum + ", " + Year

```

The regular expression for *Year* allows a range from year 1000 to 2999, forcing the first digit to be 1 or 2, and then requiring 3 consecutive digits, each in the range 0–9. The regular expression for *Month* is simply a list of possible month names. The regular expression for *DayNum* allows a range of 1 to 31, by combining 4 subranges: between 1 and 9, or between 10 and 29, or 30, or 31.

The baseline 1 for high recall only uses the regular expression for the *Year*. The baseline 2 for high precision requires a sequence of *Month* followed by a space, followed by a *Day*, a comma, another space, and a *Year* (e.g., January 5, 1999).

Table 3.7 displays the Yes/No results of these two baseline algorithms, in comparison with the gold standard defined by Judge 1 earlier. Table 3.8 shows the contingency tables for both algorithms.

Table 3.7 Results of two baseline algorithms for the extraction of DATE instances

No.	Sentence	Judge 1	Baseline 1	Baseline 2
1	He was born on the 8th of July 1987.	Yes	Yes	No
2	Cancel the flight 2000 by Sept. 10th to avoid paying fees.	No	Yes	No
3	A date to remember is August 10, 1999.	Yes	Yes	Yes
4	I wrote “10/04/1999” on the form, next to my name.	Yes	Yes	No
5	Class 20–09, that was a nice course.	No	No	No
6	The wedding was in 2008, on April 20th exactly.	Yes	Yes	No
7	Flight Air Canada 1987, is 6th in line to depart.	No	Yes	No
8	The expiration date is 15/01/12.	Yes	No	No
9	Your appointment is on October 31, 2012.	Yes	Yes	Yes
10	It is Dec. 15 already, hard to believe that 2011 is almost over.	Yes	Yes	No
11	November 1986, on the 7th day, the train departed.	Yes	Yes	No
12	There was 5000 mm of rain in 7 days.	No	No	No
13	He was in his 7th year in 1987.	No	Yes	No
14	The product code is 1111–09.	No	Yes	No
15	He arrived 31st in rank at the 2005 and 2008 runs.	No	Yes	No
16	The event happens March 10–12 2015.	Yes	Yes	No
17	The big day is November 20th, 2017	Yes	Yes	No

Table 3.8 Contingency table for baseline algorithms

		Judge 1 (Gold standard)	
		correct	incorrect
Baseline 1	correct	8	6
	incorrect	1	2
Baseline 2	correct	2	0
	incorrect	7	8

From Table 3.8, we can evaluate recall, precision, and F1 measures for both algorithms, as following:

Baseline 1—High recall	Baseline 2—High precision
$Recall = \frac{8}{9} = 88.9\%$	$Recall = \frac{2}{9} = 22.2\%$
$Precision = \frac{8}{14} = 57.1\%$	$Precision = \frac{2}{2} = 100.0\%$
$F1 = 2 * \frac{88.9 * 57.1}{88.9 + 57.1} = 69.6\%$	$F1 = 2 * \frac{22.2 * 100}{22.2 + 100} = 36.4\%$

As expected, the first baseline shows quite high recall, and the second baseline shows low recall but high precision. We now try to refine our regular expressions to maintain a high recall, but not at the expense of a low precision.

3.4.3 Refining the DATE Expressions

At this point, we will have to engage in what is essentially a process of trial and error, where we observe the data and try to find expressions that maximize both recall and precision. Both the individual elements defined earlier (the patterns for months, years, and days) and the patterns of combination can be refined to create different options. Here are some ideas:

```

Year          = [12] [0-9] {3}
Month         = (January|February|March|April|May|June|July|
                August|September|October|November|December)
MonthShort    = (Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) . *
DayNum       = ([1-9] | [1-2] [0-9] | 30|31)
DayEnd       = (1rst|2nd|3rd| [4-9]th|1[0-9]th|21rst|22nd|
                23rd|2[4-9]th|30th|31rst)
RegExDate1 = "(" + Month + "|" + MonthShort + ")" + " " +
              "(" + DayNum + "|" + DayEnd + ")" + " ", " + Year
RegExDate2 = DayNum + "/" + MonthNum + "/" + [0-9]{2}

```

How far did that refinement take us? Well, an extraction process relying on both *RegExDate1* and *RegExDate2* now finds *August 10, 1999*, *10/04/19*, *15/01/12*, and *October 31, 2012* and provides a recall of 33 %. Although this is still not high, it is higher than baseline 2, which had recall of 20 %. The precision is now at 75 %, right between baselines 1 and 2. I will leave it as an exercise for the reader to verify the precision/recall of this extraction process, and more importantly, to continue the iterative cycle of development and evaluation, as to further refine the set of regular expressions used for extracting instances of DATE.

At the end of this **iterative refinement process**, the real test for our strategy would come from gathering a completely new and unseen set of sentences, generate a new gold standard by annotating the sentences as to their mention Yes/No of a DATE, perform the extraction process on that new set of sentences, and evaluate the results

against the new gold standard. Since we used the set of sentences from Table 3.5 during the refinement cycle, we should refer to these sentences as our **development set**. It is too lenient to evaluate strategies on the development set, since that is the set on which we tried to optimize both precision and recall. The final evaluation of a strategy should be on a **test set**, never seen during the refinement cycle. It is important to understand the difference between the two sets and the variation on results which can occur, depending on which set we evaluate on. Being aware of this differentiation can help sometimes understand why results presented in research articles may vary widely, not just because of their variation in terms of strategies, but also in terms of their evaluation method.

When test sets are not available, an approximation of an algorithm's performance in test condition can be measured using a **cross-validation** approach. The idea in cross-validation is to separate the development set into N subsets, develop the rules based on $N - 1$ subsets, and test them on the N th subset, repeating such process N times and averaging the results. Cross-validation is most often optimistic in its evaluation, as it is based on a somewhat artificial setting, which in theory is valid, but where in practice, "pretending" that we have not seen part of the data either for rule development or feature development (in machine learning approaches) is not realistic. Still, cross-validation is widely used as an evaluation approach for many different categorization tasks.

3.5 Language Dependency of Named Entity Expressions

In Chap. 2, Sect. 2.6, we discussed the fact that entities themselves are referred to differently in different languages. But what about entity classes? Well, if we were to try to find information about an entity class in a multilingual setting (e.g., COMPOSER), using gazetteers of entities (e.g., *L.V. Beethoven and W.A. Mozart*), we would find ourselves once again faced with the original problem, this time generalized to all entities of a class.

When it comes to classes that are better represented by regular expressions (e.g., non-enumerable entities), there is equal potential for linguistic and/or cultural variation. When searching in different languages and/or in text written in different countries, we should be aware that this variation affects even the simplest of writing practices. Dates, for example, can be represented in many different ways, and a mention like *11/12/2016* could refer to the 11th day of the 12th month, or to the 12th day of the 11th month, depending on the country.

Dates are just one example. The writing of numbers can vary as well, with some countries using a comma to represent decimals, and others using a period (e.g., 2.54 versus 2,54). Other examples include elements of daily life in text, such as phone numbers and zip/postal codes. These too have different formats depending on the country.

If we wish to define and test regular expressions for particular entity types, and if we aspire to have them work in a multilingual setting, it is essential that we create

a gold standard that contains positive and negative sentences from the languages we intend to cover. This process might begin with writing regular expressions that are language independent, but they would eventually be refined to be language specific.

3.6 In Summary

- The term *named entity* can be interpreted in a strict sense to mean entities that can be uniquely identified, or in a looser sense to mean any identifiable information that would be of importance in text understanding.
- Different authors have proposed varying lists of named entity types, some more coarse-grained, others more fine-grained.
- Regular expressions serve as a good alternative (or complement) to lists/gazetteers, which themselves are never complete.
- Regular expressions provide a powerful way of expressing certain entity types, in order to search for them in text.
- An iterative refinement process can be used for the development of regular expressions, which aims at maximizing both recall and precision on a development dataset.
- When it comes to classification tasks, people are not always in agreement. *Cohen's Kappa* is useful for evaluating inter-annotator agreement.
- Low inter-annotator agreement can be interpreted in two ways: either the task is not well defined or the task is very subjective.
- In a multilingual context, entities are often expressed in slightly different ways. We should not assume that previously defined regular expressions are simply transferable from one language to another.

3.7 Further Reading

Entity types: Early definitions of named entities, in the Message Understanding Conference (MUC-6), are found in Sundheim (1995). An example of a fine-grained list, the *Extended Named Entity list*, is presented in Sekine et al. (2002) and further described at http://nlp.cs.nyu.edu/ene/version7_1_0Beng.html. Some examples of entity types presented in this chapter are taken from another fine-grained list used in Ling and Weld (2012).

Named Entity Recognition: A survey of NER is presented in Nadeau and Sekine (2007). In Ratnikov and Roth (2009), the emphasis is on challenges and misconceptions in NER. In Tkachenko and Simanovsky (2012), the focus is rather on features important to the NER task. The references above will at one point mention the use of gazetteers as an important element in NER.

Inter-Annotator Agreement: Cohen's kappa is presented in the Wikipedia page http://en.wikipedia.org/wiki/Cohen's_kappa which also describes some variations. The Kappa interpretation scale presented in this chapter was proposed by Koch and Landis (1977).

3.8 Exercises

Exercise 3.1 (Entity types)

- a. In Sect. 3.1, we presented a fine-grained list of entity types. For each entity type in the list (e.g., ACTOR, AIRLINE, RELIGION, and RESTAURANT), determine which of the two approaches, either gazetteers or regular expressions, would be the most appropriate in an extraction process. Discuss your choices and provide examples to support them.

Exercise 3.2 (Regular expressions)

- a. Go to the site <http://regexpal.com/>, which allows you to enter regular expressions and test them on different texts. You can also use the regular expression matcher libraries within your favourite programming language if you prefer. Make up test sentences and try the regular expressions defined in Table 3.4 to extract various type of information. Try new expressions. Play around a little to familiarize yourself with regular expressions.
- b. Write a regular expression that would be able to detect URLs. Then, write another one for emails and a third for phone numbers.
- c. Find ways to make the regular expressions in Table 3.4 more general. How would you go about this?
- d. Go back to Table 3.3 and write regular expressions to cover all the variations shown.

Exercise 3.3 (Gold standard development and inter-annotator agreement)

- a. What would be the impact of choosing negative examples for the entity DATE that do not contain any numbers? Discuss.
- b. Assuming you have access to a large corpus, what could be a good way of gathering sentence candidates for positive and negative examples for the DATE entity type?
- c. In assessing the presence of dates in the examples of Table 3.5, assume a third judge provided the following positives: {1, 3, 4, 5, 6, 8, 10, 11, 15, 16, 17}. What would be her level of agreement with the other two judges?

Exercise 3.4 (Iterative development process)

- a. Continue the development of regular expressions for finding dates in Table 3.5. Try to achieve 100 % recall and 100 % precision on that development set. Then, develop a new test set, including positive and negative examples, or better yet, to be completely unbiased, ask one of your colleagues to build that test set for you. Try the set of regular expressions you had developed. Do you still achieve good recall and precision? How much does the performance drop? Discuss.
- b. Assuming we want to extract a person's birthdate from the short abstract (in DBpedia) describing that person's life. Write a set of regular expressions which automatically extract the birth date from the abstracts for some well-known people. Develop the regular expressions in your program using a first development

set of 10 famous people and then perform a final test of your program using a new test set of 10 other famous people.

- c. Choose 5 entity types from Exercise 3.1 for which you thought regular expressions would be appropriate. Gather examples for these types from DBpedia (or another resource). For each type, gather 10 examples for your development set, and 10 examples for your test set. Use your development set to create regular expressions. Then, test your regular expressions on the test set. Do results vary depending on the entity type? Discuss.



<http://www.springer.com/978-3-319-41335-8>

Natural Language Understanding in a Semantic Web
Context

Barrière, C.

2016, XVII, 317 p., Hardcover

ISBN: 978-3-319-41335-8